UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Otimização com Muitos Objetivos por Evolução Diferencial Aplicada ao Escalonamento Dinâmico de Projeto de Software

Allan Vinicius Rezende

São Cristóvão

2019

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Allan Vinicius Rezende

Otimização com Muitos Objetivos por Evolução Diferencial Aplicada ao Escalonamento Dinâmico de Projeto de Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

Orientadora Profª. Dra. Leila Maciel de Almeida e Silva

Coorientador: Prof. Dr. André Britto de Carvalho

São Cristóvão
2019

Allan Vinicius Rezende

Otimização com Muitos Objetivos por Evolução Diferencial Aplicada ao Escalonamento Dinâmico de Projeto de Software/ Allan Vinicius Rezende. – São Cristóvão, 2019-

181 p. : il. (algumas color.) ; 30 cm.

Orientadora Profª. Dra. Leila Maciel de Almeida e Silva

Dissertação (Mestrado) – UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, 2019.

1. Escalonamento de projeto de software. 2. Search-based software engineering. 3. Otimização com muitos objetivos. 4. Evolução diferencial I. Leila Maciel de Almeida e Silva. II. Universidade Federal de Sergipe. III. Programa de Pós-Graduação em Ciência da Computação.

CDU 02:141:005.7

Allan Vinicius Rezende

Otimização com Muitos Objetivos por Evolução Diferencial Aplicada ao Escalonamento Dinâmico de Projeto de Software

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Profª. Dra. Leila Maciel de Almeida e Silva, Presidente
Universidade Federal de Sergipe (UFS)

Prof. Dr. André Britto de Carvalho, Membro
Universidade Federal de Sergipe (UFS)

Profª. Dra. Thelma Elita Colanzi Lopes, Membro
Universidade Estadual de Maringa (UEM)

Prof. Dr. Alberto Costa Neto, Membro
Universidade Federal de Sergipe (UFS)

UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
COORDENAÇÃO DE PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da Sessão Solene de Defesa da Dissertação do Curso de Mestrado em Ciência da Computação-UFS.
Candidato: ALLAN VINICIUS REZENDE

Em 05 dias do mês de Julho do ano de dois mil e dezenove, com início às 09h00min, realizou-se na Sala de Seminário do DCOMP da Universidade Federal de Sergipe, na Cidade Universitária Prof. José Aloísio de Campos, a Sessão Pública de Defesa de Dissertação de Mestrado do candidato **ALLAN VINICIUS REZENDE**, que desenvolveu o trabalho intitulado: ***" Otimização com Muitos Objetivos por Evolução Diferencial Aplicada ao Escalonamento Dinâmico de Projeto de Software"***, sob a orientação da Profª. Drª. **Leila Maciel de Almeida e Silva** e coorientação do Prof.º Drº. **André Britto de Carvalho**. A sessão foi presidida pela Profª. Drª. **Leila Maciel de Almeida e Silva** (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, a Profª. Drª. **Thelma Elita Colanzi Lopes** (Externo à Instituição / UEM), o Prof. Dr. **Alberto Costa Neto** (Externo ao Programa/ UFS), e, em seguida, o Prof. Dr. **André Britto de Carvalho** (PROCC/UFS). Após as discussões, a Banca Examinadora reuniu-se e considerou o mestrando (a) **Aprovado** *"(aprovado/reprovado)"*. Atendidas as exigências da Instrução Normativa 01/2017/PROCC, do Regimento Interno do PROCC (Resolução 67/2014/CONEPE), e da Resolução nº 25/2014/CONEPE que regulamentam a Apresentação e Defesa de Dissertação, e nada mais havendo a tratar, a Banca Examinadora elaborou esta Ata que será assinada pelos seus membros e pelo mestrando.

Cidade Universitária *"Prof. José Aloísio de Campos"*, 05 de Julho de 2019.

Profª. Drª. Leila Maciel Almeida e Silva
(PROCC/UFS)
Presidente

Prof. Dr. André Britto de Carvalho
(PROCC/UFS)
Examinador Interno

Prof. Dr. Alberto Costa Neto
(UFS)
Examinador Externo ao Programa

Profª. Drª. Thelma Elita Colanzi Lopes
(UEM)
Examinador Externo à Instituição

Alan Vinicius Rezende
Candidato

UNIVERSIDADE FEDERAL DE SERGIPE
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO - PROCC
Sala do Seminário do DCOMP – Departamento de Computação / UFS, Av. Marechal Rondon, S/N - Jardim Rosa Elze - Tel. (79) 3194-6353. CEP: 49100-000 - São Cristóvão - Sergipe - Brasil
E-mail: secretaria.pos@dcomp.ufs.br

Dedico este trabalho a minha mãe Jardelina (in memorian),
com todo o meu amor e gratidão.

Agradecimentos

Agradeço aos meus familiares e amigos, que me incentivaram durante todo o mestrado.

À minha esposa Tamires, pela compreensão e carinho.

Aos meus pais, pelos valores ensinados.

Ao STI, meu local de trabalho, pelas máquinas que possibilitaram a execução dos experimentos.

Aos meus mentores, Leila Silva, minha orientadora, e André Britto, meu coorientador, pela forma profissional e amiga com que me trataram durante o desenvolvimento deste trabalho e cuja seriedade e empenho foram fatores fundamentais para a conclusão do mesmo.

Resumo

Os problemas da Engenharia de Software geralmente envolvem problemas com vários objetivos e restrições, muitas vezes conflitantes entre si. Uma tendência para a solução destes problemas é o uso de algoritmos de busca e otimização para encontrar soluções de forma automática que balanceiem estes objetivos. Neste trabalho investigamos um problema na área de planejamento de software, a saber, o Problema de Escalonamento de Projetos de Software (em inglês, *Software Project Scheduling Problem* - SPSP), o qual visa alocar pessoas a tarefas em um projeto de software de forma a otimizar alguns objetivos, como por exemplo, o custo e a duração do projeto. Existem duas variações principais para este problema: a estática e a dinâmica. No SPSP estático o planejamento é realizado apenas no início do projeto, e os únicos objetivos a serem otimizados são o custo e a duração do projeto. O modelo dinâmico, chamado de DSPSP, considera que o ambiente de projetos de software está sujeito a incertezas, e o projeto pode precisar ser reescalonado ao longo do ciclo de desenvolvimento do software. Na abodargem dinâmica, muitos objetivos precisam ser otimizados, como o custo, a duração, a estabilidade e a robustez do cronograma, frente às mudanças que podem ocorrer durante o ciclo de desenvolvimento do projeto. O modelo dinâmico ainda é pouco explorado na literatura. Este trabalho propõe uma extensão do modelo dinâmico existente na literatura, pela consideração de mais dois eventos dinâmicos e da influência da experiência da equipe. O cerne principal do trabalho é a investigação da adequabilidade do algoritmo de otimização com muitos objetivos por evolução diferencial ao problema de escalonamento dinâmico de projetos de software, considerando o modelo proposto. Como o DSPSP envolve otimização dinâmica, seis variantes do algoritmo de evolução diferencial foram investigadas, cada uma delas contemplando uma ou mais técnicas de otimização dinâmica. O algoritmo de evolução diferencial e suas variantes foram comparados ao algoritmo evolucionário NSGA-III, também ainda não explorado para o DSPSP. Para a análise dos algoritmos investigados foi realizada uma bateria de experimentos. Os resultados sugerem que o algoritmo de evolução diferencial com técnicas de otimização dinâmica fornece melhores soluções para o DSPSP.

Palavras-chaves: escalonamento de projeto de software, search-based software engineering, evolução diferencial.

Abstract

Software Engineering problems often involve problems with many objectives and constraints, in most cases conflicting with each other. One trend toward solving these problems is the use of search and optimization algorithms to find solutions that automatically balance these objectives. In this work, we investigate a problem in the area of software planning, namely, the Software Project Scheduling Problem (SPSP), which aims to allocate people to tasks in a software project in order to optimize some objectives, such as project cost and duration. There are two main variations to this problem: static and dynamic. In static SPSP, the planning is done only at the beginning of the project, and the objectives to be optimized are project cost and duration. The dynamic model, called DSPSP, considers that the software project environment is susceptible to uncertainties, and the project may need to be rescheduled throughout the software development cycle. In dynamic approach, many objectives need to be optimized, such as cost, duration, stability and robustness of the schedule, to deal with the changes that may occur during the project development cycle. The dynamic model is still few explored in the literature. This work proposes an extension of the existing dynamic model in the literature, by considering two more dynamic events and the influence of team experience. The main focus of the work is the investigation of the suitability of the algorithm of optimization with many objectives by differential evolution to the dynamic software project scheduling problem, considering the proposed model. Since the DSPSP involves dynamic optimization, six variants of the differential evolution algorithm were investigated, each of them comprising one or more dynamic optimization techniques. The differential evolution algorithm and its variants were compared to the evolutionary algorithm NSGA-III, also not yet explored for DSPSP. For the analysis of the algorithms investigated a battery of experiments was carried out. The results suggest that the differential evolution algorithm with dynamic optimization techniques provides a better solutions for DSPSP.

Key-words: software project scheduling problem, search-based software engineering, differential evolution.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

ACO	*Ant Colony Optimization*
ASA	*Accelerated Simulated Annealing*
CPM	*Critical Path Method*
CMODE	*Cooperative Multiobjective Differential Evolution*
DE	*Differential Evolution*
DEPT	*Differential Evolution with Pareto Tournaments*
DOP	*Dynamic Optimization Problem*
DOPs	*Dynamic Optimization Problems*
DSPSP	*Dynamic Software Project Scheduling Problem*
EA	*Evolutionary Algorithm*
EBS	*Event-Based Scheduler*
GA	*Genetic Algorithm*
GDE3	*Generalized Differential Evolution 3*
HC	*Hill Climbing*
MOCell	*MultiObjective Cellular genetic algorithm*
MODPSP	*Multi-Objective Dynamic Project Scheduling Problem*
MO-FA	*Multi-Objective Firefly Algorithm*
MOEAs	*Multiobjective Optimization Evolutionary Algorithms*
de-MOEA	*ϵ-domination based Multiobjective Optimization Evolutionary Algorithm based rescheduling method*

MOP	*Multiobjective Optimization Problem*
MOTAMAQ	*Multi-Objective Two-Archive Memetic Algorithm based on Q-learning*
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
NSGA-III	*Non-dominated Sorting Genetic Algorithm III*
PAES	*Pareto Archived Evolution Strategy*
PERT	*Program Evaluation and Review Techniques*
PSO	*Particle Swarm Optimization*
PSP	*Project Scheduling Problem*
RCPSP	*Resource-Constrained Project Scheduling Problem*
SA	*Simulated Annealing*
SBSE	*Search-Based Software Engineering*
SBSPM	*Search-Based Software Project Management*
SPEA2	*Strength Pareto Evolutionary Algorithm 2*
SPM-Net	*Software Project Management net*
SPSP	*Software Project Scheduling Problem*
TPG	*Task Precedence Graph*
UML	*Unified Modeling Language*

Sumário

1 Introdução

A Engenharia de Software é uma disciplina da engenharia cujo foco é o custo-benefício do desenvolvimento de sistemas de software de alta qualidade (SOMMERVILLE, 2007). Ela pode ser definida como um conjunto de métodos, ferramentas e procedimentos que possibilitam o controle do processo de desenvolvimento de software, estabelecendo princípios para se obter software eficiente, confiável e economicamente viável (PRESSMAN; MAXIM, 2014).

A área de engenharia de software é ampla e engloba várias subáreas como, por exemplo, gerenciamento de projetos de software, engenharia de requisitos, projeto e desenvolvimento de software, teste de software, entre outras. A engenharia de software busca melhorar os processos de desenvolvimento em todas essas subáreas, seja especificando, projetando ou implementando soluções para problemas nestas diversas subáreas. Como as complexidades dos sistemas e dos problemas tendem a crescer, é necessário a automatização desses processos e a criação de novas técnicas para dar suporte aos engenheiros de software.

Muitos dos problemas que surgem nas várias subáreas da engenharia de software lidam com o balanceamento entre objetivos conflitantes, como por exemplo, o custo e a duração do projeto. Um bom projeto de engenharia de software busca desenvolver com o menor custo e a menor duração. No entanto, a redução da duração pode implicar na alocação de mais recursos e por conseguinte, aumentar o custo.

Para resolver esse tipo de problema, em que se busca minimizar mais de um objetivo conflitante, é necessária a configuração de inúmeras variáveis e isso pode superar a capacidade cognitiva do engenheiro de software. A Engenharia de Software Baseada em Busca (em inglês, *Search-Based Software Engineering* - SBSE) surgiu para resolver esses problemas da engenharia de software de forma automática, por meio da aplicação de algoritmos de busca. A SBSE é uma subárea da engenharia de software introduzida por Harman e Jones (2001). Nessa área, os problemas da engenharia de software são reformulados como problemas de busca e otimização, com o objetivo de encontrar soluções próximas do ótimo para problemas com requisitos e restrições conflitantes.

Diversas subáreas da engenharia de software compreendem problemas desta natureza. Por exemplo, na área de requisitos, o *Next Release Problem* tem como objetivo encontrar o conjunto ideal de requisitos que equilibram as solicitações dos clientes dentro das restrições de recursos (ZHANG; FINKELSTEIN; HARMAN, 2008); na área de testes de software, o problema da priorização de casos de teste busca gerar uma ordem de execução de teste ideal (MCMINN, 2004; AFZAL; TORKAR; FELDT, 2009; HARMAN; JIA; ZHANG, 2015); na área de projeto de software, o problema de atribuição de responsabilidades, visa distribuir a localização de métodos e atributos de maneira ótima entre as classes de um sistema (RÄIHÄ, 2010); na área de manutenção de software, o problema de refatoração, busca melhorar a reusabilidade, flexibilidade, entre outras características do software (KESSENTINI *et al.*, 2017); na área de gerenciamento de projeto de software, a estimativa de custo e esforço de um projeto precisam ser balanceadas (FERRUCCI; HARMAN; SARRO, 2014). Todos esses problemas podem ser transformados em problemas de otimização e ser resolvidos por meio de técnicas da SBSE, por meio de algoritmos meta-heurísticos.

Dentre os algoritmos explorados em SBSE, destacam-se os algoritmos evolucionários (em inglês, *Evolutionary Algorithm* - EA). Neste contexto, muitas variações dos EA que lidam com problemas de otimização têm surgido nas últimas décadas como o *Strength Pareto Evolutionary Algorithm 2* (SPEA2) (ZITZLER; LAUMANNS; THIELE, 2001), *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) (DEB *et al.*, 2002), NSGA-III (DEB; JAIN, 2014), entre outros. Uma variante promissora é a heurística de busca chamada Evolução Diferencial (em inglês, *Differential Evolution* - DE) que se mostrou superior em performance em diversas aplicações do mundo real. Um estudo feito por Vesterstrom e Thomsen (2004) mostrou que o DE é mais eficiente e robusto que o *Particle Swarm Optimization* (PSO) (EBERHART; KENNEDY, 1995) e EA quando aplicado a problemas com parâmetros de valores reais.

Para um gerenciamento eficaz de projeto de software, é necessário ter um bom planejamento do projeto (SOMMERVILLE, 2007). Na tarefa de planejamento de software, o cronograma visa controlar e monitorar o andamento do projeto. Sem um cronograma bem definido, é extremamente difícil atingir metas de custo e duração se o projeto for muito complexo (PRESSMAN; MAXIM, 2014). Na China, mais de 40% dos projetos falham devido ao planejamento ineficiente (CHEN; ZHANG, 2013). Em 2015, um relatório (The

Standish Group International, 2015) mostra que, de 50.000 projetos ao redor do mundo, apenas 29% foram bem-sucedidos, e no caso de grandes projetos, a taxa de sucesso é de 2%.

No contexto de SBSE, o problema de planejamento de software foi modelado como um problema de otimização e o cronograma do projeto é estabelecido pelo uso de técnicas de busca (ALBA; CHICANO, 2007; FERRUCCI; HARMAN; SARRO, 2014). Entretanto, o gerenciamento de projetos de software baseado em busca (em inglês, *Search-Based Software Project Management* - SBSPM) já era pesquisado antes mesmo da introdução do termo SBSE por Harman e Jones (2001). Em (CHANG *et al.*, 1998), por exemplo, os autores trataram do problema de escalonamento e alocação de recursos em projetos usando abordagens de busca e otimização.

No contexto de planejamento de software, o problema do escalonamento de projeto de software (em inglês, *Software Project Scheduling Problem* - SPSP) consiste em definir quais empregados realizarão cada tarefa do projeto e quando essas tarefas deverão ser executadas. Diferente da abordagem de Chang *et al.* (1998), o SPSP considera um único recurso no projeto, que são os empregados. Alba e Chicano (2007) definiram um modelo para esse problema, cujo objetivo é obter um cronograma que minimize dois objetivos, a duração e o custo do projeto, respeitando um conjunto de restrições. Apesar desse modelo ser bastante relevante, e estar continuamente sendo investigado (FERRUCCI; HARMAN; SARRO, 2014), não representa um modelo robusto para a realidade, pois ele considera um ambiente de projetos de software estático, o que difere da realidade onde incertezas e eventos imprevistos afetam o planejamento inicial do software. No modelo de Alba e Chicano (2007) o planejamento é realizado apenas uma vez, no início do projeto.

Nos últimos anos, trabalhos como (XIAO *et al.*, 2010) e (SHEN *et al.*, 2016), passaram a abordar modelos dinâmicos, chamados de *Dynamic Software Project Scheduling Problem* (DSPSP). Estes modelos são extensões do modelo estático de Alba e Chicano (2007) e consideram o escalonamento e atribuição das tarefas em ambientes incertos e mutáveis ao longo do tempo. Em modelos dinâmicos, o cronograma é reescalonado à medida que eventos ou incertezas que possam afetar o planejamento ocorrem. Uma característica explorada por Shen *et al* (2016) é considerar um maior número de funções objetivo, no caso quatro. Problemas de otimização com mais de três funções

objetivo, recebem uma nova denominação, chamada de otimização com muitos objetivos (SCHUTZE; LARA; COELLO, 2011; ISHIBUCHI *et al.*, 2011).

Apesar dos avanços nas pesquisas sobre esse problema, o modelo dinâmico ainda é recente e pouco explorado pelos estudos encontrados na literatura na elaboração de algoritmos dedicados ao mesmo. Além da proposição do modelo dinâmico, Shen *et al.* (2016) também propuseram um algoritmo evolutivo adaptado para o modelo, chamado de dε-MOEA. Porém, eles não avaliaram este algoritmo frente a outras meta-heurísticas. Além disso, este algoritmo não usa técnicas específicas aplicadas à resolução de problemas dinâmicos, como por exemplo, a introdução de diversidade quando mudanças ocorrerem (NGUYEN; YANG; BRANKE, 2012). Recentemente, Shen *et al* (2018) propuseram outro algoritmo para lidar com o modelo dinâmico, chamado de MOTAMAQ (SHEN *et al.*, 2018).

Em geral, as abordagens para o modelo do DSPSP não modelam o problema com técnicas para muitos objetivos, nem com técnicas para problemas dinâmicos. Além disso, não fazem uma comparação com algoritmos para muitos objetivos, como por exemplo o NSGA-III (DEB; JAIN, 2014). Extensões do modelo, permitindo uma maior aproximação com o mundo real também se fazem necessárias, como por exemplo, considerar o nível de experiência dos empregados, pois isto pode influenciar na duração da tarefa, e o tamanho da equipe que realiza a tarefa, pois isto pode minimizar a sobrecarga de comunicação, entre outros. Além disso, embora o DE tenha se mostrado promissor quando aplicado em problemas que usam números reais, como em (VESTERSTROM; THOMSEN, 2004), não foi investigado dentro do contexto do DSPSP.

1.1 Objetivos

O objetivo geral deste trabalho é investigar a adequação do algoritmo de otimização de evolução diferencial com muitos objetivos aplicado ao problema de escalonamento dinâmico de projetos de software em uma extensão do modelo dinâmico proposto por (SHEN *et al.*, 2016).

De forma a alcançar o objetivo geral, os seguintes objetivos específicos foram definidos:

1. Definir uma extensão do modelo de Shen *et al* (2016), considerando mais objetivos;
2. Elaborar e implementar o algoritmo de evolução diferencial com muitos objetivos para formulação dinâmica proposta;
3. Analisar diferentes técnicas de otimização dinâmica que lidem com as restrições dos modelos dinâmicos do DSPSP;
4. Validar o algoritmo proposto por meio de experimentos considerando diferentes instâncias do problema;
5. Comparar os resultados obtidos com os obtidos pelo NSGA-III.

1.2 Metodologia

Inicialmente foi realizada uma revisão sistemática da literatura (REZENDE *et al.*, 2019) sobre o problema de escalonamento de projetos de software, centrada no contexto da engenharia de software baseada em busca. Nesta revisão, alguns aspectos específicos das abordagens utilizadas para resolver o problema foram discutidos, como também algumas tendências e oportunidades foram identificadas para guiar pesquisas futuras na área. Além disso, características comuns das abordagens do problema foram apresentadas. Na revisão, os artigos foram classificados de acordo com um plano proposto para responder às questões de pesquisa relacionadas aos elementos principais do SPSP, como modelo utilizado, técnicas baseadas em busca, instâncias utilizadas para validar a solução e aspectos de avaliação dos resultados.

Em seguida, foi desenvolvida uma extensão para o modelo de escalonamento dinâmico de projetos de software, baseado no trabalho de Shen *et al.* (2016). A extensão do modelo busca incorporar mais características de projetos de software reais ao modelo de Shen *et al.* (2016), levando em consideração a ocorrência de eventos paralelos, além da incorporação de mais um objetivo e mais eventos dinâmicos, como a remoção de tarefas e a chegada de um novo empregado.

Após a definição do modelo estendido, foi implementada uma extensão do *framework* desenvolvido por Amaral (2018) para o modelo de Shen *et al.* (2016), visando incorporar a extensão do modelo aqui proposto.

O passo seguinte foi a implementação de um algoritmo de evolução diferencial cooperativo com múltiplas populações para otimizações multiobjetivo, CMODE, que permite explorar algumas características dinâmicas do problema. O algoritmo foi baseado no trabalho de Wang, Zhang e Zhang (2016). Além disso, algumas variantes do CMODE incorporando técnicas de otimização dinâmica foram desenvolvidas. Nosso trabalho avaliou dois algoritmos que ainda não haviam sido explorados nos estudos sobre o DSPSP: CMODE e NSGA-III.

Para avaliar o desempenho dos algoritmos, fez-se necessário modificar o gerador de instâncias utilizados por Shen *et al.* (2016) para contemplar as modificações referentes à extensão do modelo proposto neste trabalho. Após a modificação e geração das instâncias, o *framework* estendido foi utilizado para conduzir os experimentos. Após a realização das simulações, os resultados foram medidos e comparados em termos das métricas do hipervolume, acurácia e estabilidade das soluções obtidas (HELBIG; ENGELBRECHT, 2013). Na avaliação dos resultados, o desempenho do algoritmo proposto foi comparado com o NSGA-III. A comparação de desempenho do CMODE e NSGA-III inclui a avaliação do impacto da incorporação das técnicas de otimização dinâmica nestes algoritmos.

1.3 Contribuições deste trabalho

Esta dissertação visa contribuir nos seguintes itens:

- Extensão do modelo do DSPSP proposto por Shen *et al.* (2016), para deixá-lo mais próximo de projetos do mundo real, resultando na adição de mais um objetivo ao problema, além de considerar mais eventos dinâmicos;
- Extensão do framework para o experimento do problema de escalonamento de projeto de software, inicialmente desenvolvido por Amaral (2018) para o modelo de Shen *et al.* (2016);
- Incorporação de técnicas de otimização dinâmica ao algoritmo de evolução diferencial com muitos objetivos e sua aplicação ao problema do DSPSP;

- Incorporação de técnicas de otimização dinâmica ao NSGA-III e aplicação do algoritmo no problema do DSPSP;
- Análise comparativa dos resultados obtidos pela aplicação do CMODE e NSGA-III, ambos abordados pela primeira vez para o DSPSP.

Além disso, um resultado complementar foi a realização de uma revisão sistemática da literatura para o SPSP (REZENDE *et al.*, 2019), onde sumarizamos os principais aspectos relacionados ao problema.

1.4 Estrutura do documento

A dissertação está organizada como descrito a seguir. O capítulo 2 apresenta os principais conceitos necessários ao entendimento do trabalho, bem como os principais trabalhos relacionados. No capítulo 3 descreve-se o modelo proposto para o problema do escalonamento de projeto de software e o algoritmo usado para resolvê-lo. Em seguida, no capítulo 4, são descritos os experimentos realizados e a discussão destes resultados. Por último, o capítulo 5 apresenta algumas considerações finais e direções de trabalhos futuros.

2 Referencial Teórico

Neste capítulo são apresentados os principais conceitos para a compreensão deste trabalho. Primeiramente é apresentada uma visão geral da área da engenharia de software baseada em busca, posteriormente o problema de escalonamento de projetos de software é descrito considerando o contexto da SBSE. Em seguida, são apresentados as principais técnicas e conceitos envolvidos na otimização multiobjetivo, incluindo princípios da otimização dinâmica, bem como as meta-heurísticas de evolução diferencial e o NSGA-III, abordados neste trabalho. Por fim, alguns trabalhos relacionados são brevemente descritos.

2.1 Engenharia de software baseada em busca

A engenharia de software baseada em busca foi introduzida por Harman e Jones (2001), como um novo campo de pesquisa na engenharia de software, no qual problemas clássicos da engenharia de software podem ser redefinidos como problemas de otimização. Na SBSE o termo "busca" está relacionado às técnicas de busca e uma visão geral da área de SBSE pode ser encontrados em (HARMAN; MANSOURI; ZHANG, 2012) e (HARMAN *et al.*, 2012).

Uma característica comum as diversas subáreas da engenharia é a necessidade de otimizar o equilíbrio entre o uso de recursos e o desempenho da solução proposta. Dessa forma, uma das preocupações da engenharia de software é tratar de problemas com essas mesmas características, buscando-se construir sistemas que sejam mais rápidos, baratos, confiáveis, escaláveis e adaptáveis, entre outros objetivos possíveis (FERRUCCI; HARMAN; SARRO, 2014). De modo geral, esses objetivos são conflitantes entre si, e soluções ótimas são difíceis de encontrar (HARMAN; JONES, 2001). Portanto, técnicas baseadas em busca são boas candidatas para resolver esses tipos de problemas. Essas técnicas podem incluir programação linear, para soluções exatas, ou meta-heurísticas, como algoritmos genéticos (COLANZI *et al.*, 2013).

Algoritmos de otimização baseado em busca são usados na SBSE para identificar

soluções ótimas ou quase ótimas. O espaço de busca é o espaço de todas as soluções candidatas e é geralmente enorme, sendo difícil ser explorado completamente. Para resolver um problema no contexto da engenharia de software baseada em busca, é preciso definir uma representação do problema, como também a função objetivo para guiar o procedimento de busca, comparando a qualidade das soluções obtidas durante o processo de busca. Então, algoritmos de otimização baseados em busca são usados para encontrar soluções ótimas ou quase ótimas para o problema. Muitas heurísticas foram propostas: técnicas de busca local, como *Hill Climbing* (HC); e técnicas de busca global, como algoritmos genéticos (em inglês, *Genetic Algorithm* - GA) (LUKE, 2013).

A área da SBSE contempla diversos domínios, diretamente ligados aos problemas da engenharia de software de que tratam. Alguns dos domínios comumente tratados na literatura são: requisitos (ZHANG; FINKELSTEIN; HARMAN, 2008), testes (MCMINN, 2004; AFZAL; TORKAR; FELDT, 2009; HARMAN; JIA; ZHANG, 2015), projeto de software (RÄIHÄ, 2010), gerência (FERRUCCI; HARMAN; SARRO, 2014) e distribuição, manutenção e melhoria de software (KESSENTINI *et al.*, 2017). Este trabalho insere-se na subárea da SBSE chamada de gerenciamento de projeto de software baseado em busca (SBSPM) que se enquadra no domínio de gerência de projetos.

2.2 Problema de escalonamento de projeto de software

Dentro da área de gerência de projeto de software, existe a área de planejamento de software. Para um gerenciamento eficaz de projeto de software, ter um bom planejamento do projeto se faz necessário (SOMMERVILLE, 2007), balanceando duração e custos do projeto. Além disso, em projetos de médio a grande porte, o planejamento do projeto é muito complexo e desafiador (DUGGAN; BYRNE; LYONS, 2004), porque os gerentes têm que lidar com objetivos conflitantes como tempo e qualidade, e uma grande variação de produtividade entre os desenvolvedores. A tarefa de planejar o software requer que o gerente de projeto estime o custo do projeto, defina o cronograma do projeto e a alocação de recursos. Para estimativa de custo, modelos como COCOMO-II (*Constructive Cost Model II*) (BOEHM *et al.*, 2000) têm sido desenvolvidos e utilizados. Para o cronograma, o gerenciamento é geralmente conduzido por ferramentas e técnicas, como PERT (*Program Evaluation and Review Techniques*) (PMI, 2013), o CPM (*Critical*

Path Method) (SHTUB *et al.*, 2005) e o problema de escalonamento de projeto com restrição de recurso (em inglês, *Resource-Constrained Project Scheduling Problem* - RCPSP) (BRUCKER *et al.*, 1999). O objetivo do cronograma é controlar e monitorar o progresso do projeto. Sem um cronograma bem definido, é extremamente difícil alcançar os objetivos de custo e duração se o projeto for muito complexo.

O problema de escalonamento de projetos de software é um dos problemas mais significativos da área de planejamento de software e compreende duas atividades principais: (*i*) definição de quais empregados devem ser alocados para executar cada tarefa do projeto, e (*ii*) determinação do tempo de que cada tarefa deve ser executada. Alba e Chicano (2007) definiram uma das mais conhecidas formulações para esse problema, chamada de problema de escalonamento de projeto (em inglês, *Project Scheduling Problem* - PSP). Relacionada com projeto de software, alguns autores chamam esse problema de problema de escalonamento de projeto de software (SPSP, ou SPS *problem*), como (LUNA *et al.*, 2014), nomeclatura esta adotada nesse trabalho.

O SPSP pode ser considerado como uma variação do problema de escalonamento de projeto com restrição de recurso (RCPSP), uma vez que ambos tentam obter um cronograma que minimize a duração do projeto, de acordo com um conjunto de restrições, e os dois podem usar meta-heurísticas para encontrar boas soluções (ALBA; CHICANO, 2007). Entretanto, o SPSP também considera como um objetivo adicional minimizar o custo do projeto. Além disso, os empregados são os únicos recursos no SPSP. Cada empregado pode ter várias habilidades e ele/ela pode realizar várias tarefas em um dia de trabalho (ALBA; CHICANO, 2007). Os empregados recebem um salário para executar as tarefas do projeto.

Na maioria dos projetos, as tarefas podem incluir uma relação de precedência de tal maneira que uma tarefa τ_i precise ser concluída antes que uma tarefa τ_j comece. Cada empregado pertencente à equipe do projeto é associado a pelo menos uma tarefa. A Figura 2.1 mostra um exemplo de uma equipe do projeto. Os atributos de um empregado e_i são: o conjunto de habilidades, $e_i^{habilidades}$; a dedicação máxima ao projeto, e_i^{maxded}; e o valor do seu salário mensal, $e_i^{salario}$. Por exemplo, um empregado pode ter uma grande experiência em *Unified Modeling Language* (UML) e banco de dados, enquanto outro pode ser um especialista em programação e com bom conhecimento em *web design*. A dedicação máxima de um empregado no projeto é definida pela quantidade de horas

dedicadas para o projeto durante um dia de trabalho.

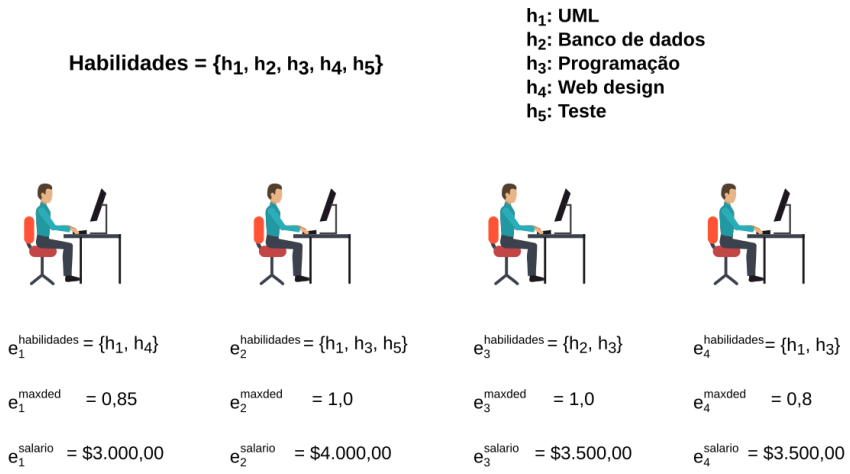

Figura 2.1 – Exemplo de uma equipe de empregados do projeto (adaptado de (ALBA; CHICANO, 2007)

As tarefas do projeto são unidades ou pacotes de trabalho que são bem delimitadas (Figura 2.2). Cada tarefa τ_j, requer um conjunto de habilidades necessárias para ser realizada, $\tau_j^{habilidades}$ e um esforço estimado, $\tau_j^{esforco}$, expresso em homem-mês, que define o tempo que um único empregado levaria para completar a tarefa (ALBA; CHICANO, 2007).

A relação entre tarefas é modelada em (ALBA; CHICANO, 2007) como um grafo de precedência de tarefas (em inglês, *Task Precedence Graph* - TPG). O TPG é um grafo acíclico direcionado $G(T, A)$ com um conjunto de vértices T que representa o conjunto das tarefas e A um conjunto de arestas que representam as relações de dependências entre tarefas. Uma aresta $(\tau_i, \tau_j) \in A$ indica que a tarefa τ_i precisar ser finalizada antes que a tarefa τ_j possa ser iniciada.

O SPSP visa minimizar dois objetivos do projeto: duração e custo. Além disso, algumas restrições são consideradas: cada tarefa precisa ser executada por pelo menos um empregado; o conjunto de habilidades requeridas por uma tarefa deve pertencer ao conjunto da união das habilidades dos empregados alocados para esta tarefa; e nenhum empregado deve exceder sua dedicação máxima ao projeto.

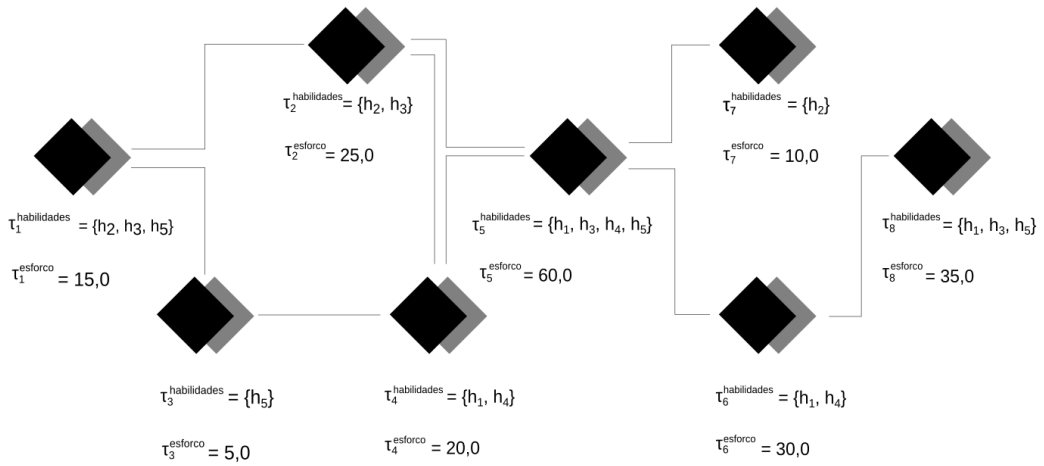

Figura 2.2 – Exemplo de um grafo de precedência de tarefas de um projeto

A solução para o problema é representada por uma matriz $MD = (md_{ij})$ de tamanho $|E| \times |\mathrm{T}|$, onde $md_{ij} \geq 0$, $1 \leq i \leq |E|$, $1 \leq j \leq |\mathrm{T}|$ e E representa o conjunto dos empregados. Cada elemento md_{ij} indica o grau de dedicação do empregado e_i para a tarefa τ_j. Portanto, se o empregado e_i executar a tarefa τ_j com um grau de dedicação de 0,8, isso significa que ele/ela ocupará 80% do seu expediente de trabalho nesta tarefa. Se o grau de dedicação for 0, então o empregado não está alocado para a tarefa. A Figura 2.3 mostra um exemplo de uma solução candidata do SPSP na forma de uma matriz de dedicação.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
e_1	0,00	0,00	0,00	0,80	0,40	0,20	0,00	0,50
e_2	1,00	0,25	0,75	0,25	0,80	0,30	0,00	0,35
e_3	0,50	0,60	0,00	0,00	1,00	0,00	1,00	0,40
e_4	0,75	0,45	0,00	0,60	0,30	0,75	0,00	0,50

Figura 2.3 – Exemplo de uma matriz de dedicação

A informação da matriz de dedicação que é apresentada na Figura 2.3 é usada para calcular a duração de cada tarefa, como também o tempo de início e fim da mesma (ALBA; CHICANO, 2007). Para calcular a duração do projeto, primeiro a duração de cada tarefa é calculada por:

$$\tau_j^{dur} = \frac{\tau_j^{esforco}}{\sum_{i=1}^{|E|} md_{ij}}$$

Depois disso, o TPG é usado para obter a duração do projeto. O objetivo final é construir o diagrama de *Gantt* (PMI, 2013), expressando o cronograma do projeto (Figura 2.4).

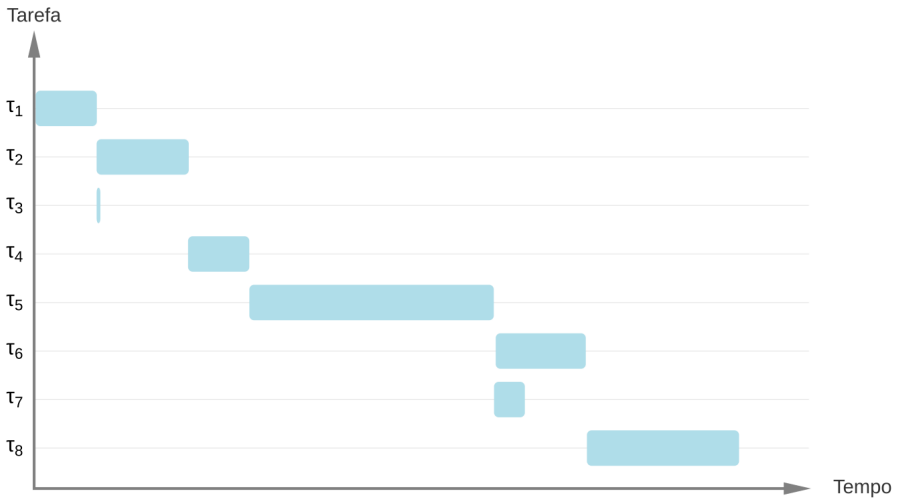

Figura 2.4 – Exemplo do diagrama de *Gantt*

O custo do projeto é a soma do pagamento dos empregados pela dedicação de cada um ao projeto. Este custo é obtido multiplicando os salários dos empregados pelo tempo gasto por eles no projeto. O tempo, por sua vez, é a soma das dedicações multiplicado pela duração de cada tarefa (ALBA; CHICANO, 2007).

$$P_{custo} = \sum_{i=1}^{|E|} \sum_{j=1}^{|T|} e_i^{salario} \cdot md_{ij} \cdot \tau_j^{dur}$$

2.2.1 Escalonamento dinâmico de projetos de software

O modelo do SPSP, apresentado na Seção 2.2 é baseado no fato de que as informações referentes às tarefas identificadas, como dependências entre tarefas e esforço estimado, não mudam no decorrer do tempo. Pressupõe-se que essas informações são conhecidas e permanecem inalteradas durante todo o curso do projeto. Sendo assim, esse modelo é classificado como estático.

Entretanto, os projetos do mundo real estão sujeitos às incertezas que não puderam ser previstas inicialmente. Devido à flexibilidade do software, projetos de desenvol-

vimento e manutenção estão sujeitos a mudanças, uma vez que mudanças dos requisitos por parte dos clientes é algo normal.

Considerando esse fato, Xiao *et al.* (2010) realizaram um trabalho onde consideram situações disruptivas que podem levar ao replanejamento do cronograma, propondo um método multiobjetivo para calcular os replanejamentos por meio de algoritmos genéticos. Posteriormente, Shen *et al.* (2016) estenderam o modelo do SPSP estático e consideraram as incertezas e eventos dinâmicos que ocorrem com frequência durante o desenvolvimento de projetos de software, de maneira que seja possível replanejar o cronograma do projeto. Para contemplar essa formulação, foi elaborado um modelo matemático para o problema multiobjetivo de escalonamento dinâmico de projeto, chamado MODPSP (*Multi-Objective Dynamic Project Scheduling Problem*) (SHEN *et al.*, 2016). Nesse modelo, quatro objetivos são considerados para a solução do problema: custo do projeto, duração do projeto, robustez do cronograma, e estabilidade do cronograma.

Com esse novo modelo, dois novos objetivos foram adicionados em relação ao modelo estático do SPSP, descrito por Alba e Chicano (2007). A robustez expressa a sensibilidade da qualidade de um cronograma em relação às incertezas das estimativas de esforço da tarefa, e a estabilidade mede o quanto um cronograma novo difere em relação ao cronograma inicial.

O modelo dinâmico de Shen *et al.* (2016) faz uso de um método baseado em cenários, que consiste em gerar um conjunto $\{\theta_q | q = 1, 2, ..., N\}$ de cenários aleatórios de duração e custo para as tarefas do projeto, onde θ_q é a q-ésima amostra do cenário de esforço para a tarefa e N é o tamanho da amostra. Esses cenários são utilizados para o cálculo da robustez do cronograma em um dado tempo t_l.

2.3 Representação do DSPSP

Essa seção detalha os principais elementos do DSPSP de acordo com a modelagem utilizada por Shen *et al.* (2016).

2.3.1 Incorporação de incertezas

Esse modelo leva em consideração a incerteza inerente a qualquer estimativa de esforço para realização das tarefas. No início do projeto o esforço de cada tarefa pode ser estimado com algum método, como por exemplo o COCOMO (BOEHM *et al.*, 2000). Entretanto, modificações na especificação ou imprecisões na estimativa inicial podem causar mudanças na estimativa inicial de esforço das tarefas. Sendo assim, considera-se que o valor do esforço segue uma distribuição normal $\phi(\mu, \sigma)$, de modo que as tarefas possuam esforço estimado médio μ e desvio padrão σ. Dessa forma, no início do projeto assume-se o valor μ para o esforço estimado. Porém, a cada vez que um evento dinâmico ocorre, aproveita-se o reescalonamento necessário para realizar uma nova estimativa de esforço restante para cada tarefa. O novo valor é obtido escolhendo aleatoriamente um valor da distribuição normal $\phi(\mu, \sigma)$, e em seguida subtrai-se o esforço já realizado.

2.3.2 Incorporação de eventos dinâmicos

Além da incerteza na estimativa de esforço, a principal diferença do DSPSP para o SPSP é a ocorrência de eventos dinâmicos durante a execução do projeto. Esses eventos têm impacto direto no cronograma do projeto, pois lidam com a adição de novas tarefas e saída/retorno de empregados. Os eventos considerados no modelo são:

(1) **Chegada de novas tarefas**: No decorrer do projeto é comum surgirem novos requisitos. Com isso, novas tarefas podem ser incluídas a qualquer tempo. Esse dinamismo é muito comum em projetos grandes e complexos, onde os requisitos tendem a ser altamente voláteis e mutáveis durante o tempo de vida do projeto. As tarefas podem ser classificadas como urgentes ou regulares. Tarefas urgentes são aquelas que precisam ser executadas imediatamente, enquanto tarefas regulares podem ser agendadas para o momento adequado, respeitando-se as dependências entre as tarefas já existentes.

(2) **Saída de empregados**: Empregados podem precisar deixar o projeto temporariamente por motivo de doença ou de participação em outros projetos ou por outras razões. Nesse caso, o cronograma precisa ser adaptado para distribuir as tarefas em execução entre os empregados que continuam no projeto.

(3) **Retorno de empregados**: Após a saída do empregado, ele/ela pode retornar ao projeto. Sendo assim, um empregado que tenha saído do projeto pode retornar a qualquer tempo, ficando disponível para ser alocado nas tarefas.

2.3.3 Propriedades dos empregados

Suponha que o projeto requer um conjunto H de habilidades e que existe um conjunto E de empregados. Seja t_l, $l = 0, 1, 2, ...$, um determinado instante em que algum tipo de evento dinâmico ocorre (incluindo o tempo inicial t_0), no qual é necessário executar um método de reescalonamento. Cada empregado e_i, $i = 1, 2, ..., |E|$, possui algumas variáveis ($e_i^{hab}, hab_i, e_i^{maxded}, e_i^{salario}, e_i^{salario\_extra}, e_{ij}^{proficiencia}$) que são consideradas fixas durante todo o projeto. No entanto, durante o projeto um empregado e_i pode sair ou retornar ao projeto em um dado instante t_l. A variável binária $e_i^{disponivel}(t_l)$ indica se o empregado e_i pode ou não ser alocado a alguma tarefa no instante t_l. A Tabela 2.1 descreve essas variáveis.

Tabela 2.1 – Variáveis dos empregados

Variáveis	Descrição
e_i^{hab}	Conjunto de valores que indica o quanto o empregado e_i domina cada uma das habilidades contidas em H.
hab_i	Conjunto de habilidades nas quais o empregado e_i possui algum nível de proficiência.
e_i^{maxded}	Valor de dedicação máxima do empregado e_i ao projeto. É o percentual do expediente normal de um empregado. O valor $e_i^{maxded} = 1,1$ indica que e_i é permitido trabalhar 110% do tempo normal de trabalho. Um empregado pode ter dedicação total ($e_i^{maxded} = 1$) ou parcial ($e_i^{maxded} < 1$).
$e_i^{salario}$	Valor do salário mensal de e_i referente ao expediente regular.
$e_i^{salario\_extra}$	Valor do salário mensal de e_i referente a horas extras.
$e_i^{disponivel}(t_l)$	Variável binária que indica se e_i está disponível no instante t_l.
$e_{ij}^{proficiencia}$	Nível de proficiência de e_i na tarefa τ_j.

2.3.4 Propriedades das tarefas

No tempo inicial t_0, supõe-se que existem N_I tarefas no projeto. No decorrer do projeto, novas tarefas podem ser adicionadas. Sejam $N_{novas}(t_l)$ as novas tare-

fas que chegaram no tempo t_l. Sendo assim, um total de $(N_I + N_{novas}(t_l))$ tarefas são consideradas como parte do projeto. Cada tarefa τ_j, $j = 1, 2, ..., N_I + N_{novas}(t_l)$, tem algumas variáveis $(\tau_j^{hab}, req_j, \tau_j^{esf\_tot\_est})$ que são consideradas fixas. No tempo t_l, é possível que certa tarefa tenha sido finalizada, ou que a tarefa não possa ser realizada temporariamente devido à saída de um empregado. Sendo assim, algumas variáveis $(\tau_j^{incompleta}(t_l), \tau_j^{disponivel}(t_l), \tau_j^{esf\_res\_est}(t_l))$ dependem do tempo em que os eventos ocorrem. Além disso, o TPG, que representa o grafo de precedência de tarefas, também pode sofrer variação no tempo t_l. A descrição dessas variáveis é mostrada na Tabela 2.2.

Tabela 2.2 – Variáveis das tarefas

Variáveis	Descrição
τ_j^{hab}	Conjunto de valores que indica se uma habilidade é requerida pela tarefa τ_j.
req_j	Conjunto de habilidades requeridas pela tarefa τ_j.
$\tau_j^{esf\_tot\_est}$	Esforço total estimado no início do projeto para completar a tarefa τ_j.
$\tau_j^{incompleta}(t_l)$	Variável binária que indica se a tarefa τ_j terminou no tempo t_l. $\tau_j^{incompleta}(t_l) = 1$ indica que τ_j está incompleta e $\tau_j^{incompleta}(t_l) = 0$ indica que τ_j foi finalizada.
$TPG(t_l)$	Grafo de precedência de tarefas, que pode ser atualizado a cada instante t_l caso uma nova tarefa tenha chegado ou uma tarefa tenha sido finalizada.
$\tau_j^{disponivel}(t_l)$	Variável binária que indica se τ_j está disponível no instante t_l. $(\tau_j^{disponivel}(t_l) = 1)$ ou não $(\tau_j^{disponivel}(t_l) = 0)$. A tarefa τ_j é considerada disponível em t_l se e somente se três condições forem satisfeitas simultaneamente: (i) τ_j está incompleta $(\tau_j^{incompleta}(t_l) = 1)$; (ii) para qualquer habilidade requerida por τ_j, tem pelo menos um empregado disponível em t_l que possui esta habilidade; (iii) todas as tarefas incompletas que precedem τ_j no $TPG(t_l)$ satisfazem a condição anterior.
$\tau_j^{esf\_res\_est}(t_l)$	Esforço restante estimado da tarefa τ_j no instante t_l.

O TPG é um grafo que representa os relacionamentos de dependência entre as tarefas. Pode ser atualizado em qualquer instante t_l da execução do projeto para incluir novas tarefas ou remover as tarefas já finalizadas. O processo de atualização do TPG consiste em remover os vértices das tarefas quando estas são concluídas e adicionar novos vértices quando novas tarefas chegam. Caso a tarefa que chegue seja regular,

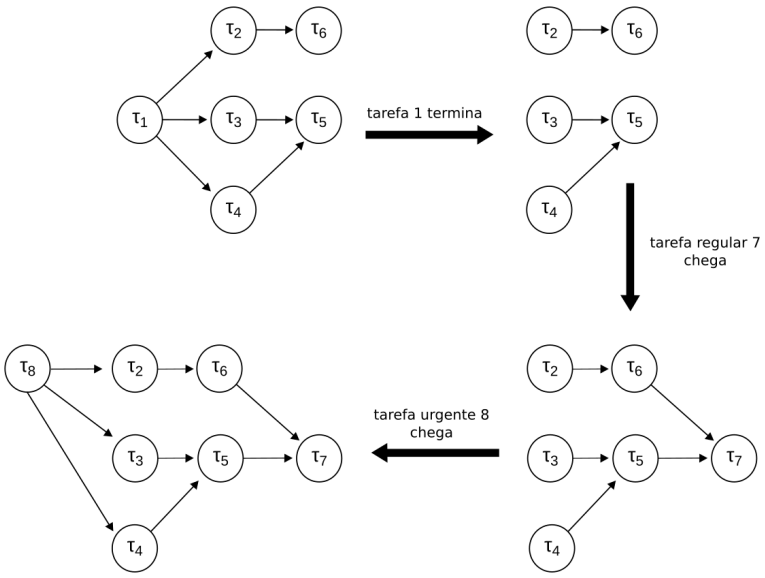

Figura 2.5 – Exemplo de atualização do grafo de precedência de tarefas (TPG)

esta é inserida após uma das tarefas incompletas. Caso a tarefa seja urgente, é inserida antes de uma ou mais tarefas incompletas, fazendo com que estas tenham sua execução interrompida até que a nova tarefa urgente seja concluída. A Figura 2.5 ilustra alguns exemplos de atualização do TPG quando uma tarefa regular ou urgente chega. Nesse caso, a tarefa τ_7 regular chega com menor prioridade e é adicionada no final do projeto, enquanto a tarefa τ_8 é urgente e é inserida no início do projeto, sem precedentes.

2.3.5 Representação da solução

As soluções do DSPSP são representadas na forma de uma matriz $MD = md_{ij}$, assim como na definição de Alba e Chicano (2007) e também adotada em Shen *et al.* (2016). A cada reescalonamento, busca-se encontrar uma nova matriz de dedicação $MD(t_l)$ no qual o valor $md_{ij}(t_l)$ indica o percentual do expediente normal que um empregado e_i dedica à tarefa τ_j no instante t_l. Como existem eventos que afetam a disponibilidade dos empregados, $md_{ij}(t_l) = 0$ para todo $j = 1, 2, ..., |\mathrm{T}|$, se $e_i^{disponivel}(t_l) = 0$. Analogamente, $md_{ij}(t_l) = 0$ para todo $i = 1, 2, ..., |E|$ quando $\tau_j^{disponivel}(t_l) = 0$. Sendo assim, apenas os valores de $md_{ij}(t_l) \in \{md_{ij}(t_l) | e_i^{disponivel}(t_l) = 1 \wedge \tau_j^{disponivel}(t_l) = 1\}$ devem ser levados em

consideração pelo método de otimização.

2.3.6 Objetivos a serem otimizados

A otimização dos objetivos é feita de acordo com o modelo de Shen *et al.* (2016), considerando que em um dado instante t_l ($t_l > t_0$), pode-se obter as seguintes informações do projeto de software:

- Um conjunto de empregados disponíveis $E\_disp(t_l)$;
- Um conjunto de tarefas disponíveis T$_disp(t_l)$ com o esforço estimado remanescente da tarefa. Para cada tarefa $\tau_j \in$ T$_disp(t_l)$, o esforço realizado de t_0 até t_l é registrado como $\tau_j^{esf\_rea}(t_l)$. Assim, o esforço restante estimado da tarefa τ_j em t_l é calculado como $\tau_j^{esf\_res\_est}(t_l) = \tau_j^{esf\_tot\_est} - \tau_j^{esf\_rea}(t_l)$;
- O $TPG(t_l) = G(\mathrm{T}(t_l), A(t_l))$ atualizado em t_l.

Um novo cronograma é gerado otimizando as funções objetivo sumarizadas na Tabela 2.3.

Tabela 2.3 – Objetivos a serem otimizados (minimizados)

Função	Objetivo	Descrição
$f_1(t_l)$	Duração	Tempo decorrido entre o instante t_l e o encerramento do projeto.
$f_2(t_l)$	Custo	Valor a ser pago aos empregados entre o instante t_l e o encerramento do projeto.
$f_3(t_l)$	Robustez	Medida do quanto o cronograma é suscetível a incertezas na estimativa de esforço. Quanto menor, melhor o desempenho.
$f_4(t_l)$	Estabilidade	Medida do quanto o cronograma em t_l difere do cronograma anterior.

A duração representa o tempo necessário para que todas as tarefas restantes no instante t_l sejam concluídas com os recursos disponíveis no momento, dada por:

$$f_1(t_l) = duracao_I = \max_{\{j|\tau_j \in \mathrm{T}\_disp(t_l)\}} (\tau_j^{fim}(t_l)) - \min_{\{j|\tau_j \in \mathrm{T}\_disp(t_l)\}} (\tau_j^{inicio}(t_l))$$

onde o subscrito I se refere ao cenário inicial, no qual considera-se que não houve variações no esforço da tarefa e considera-se o esforço estimado restante $\tau_j^{esf\_res\_est}(t_l)$ calculado acima, como o esforço restante exato da tarefa τ_j no instante t_l. Para cada uma das tarefas disponíveis τ_j em t_l, é considerado o esforço restante, descartando o esforço já realizado até o momento. Assim, $\tau_j^{inicio}(t_l)$ representa o tempo no qual o esforço remanescente da tarefa τ_j começa a ser executada depois de t_l de acordo com o novo cronograma gerado. Então temos que $\tau_j^{inicio} \geq t_l$. Já $\tau_j^{fim}(t_l)$ representa o momento em que τ_j é concluída. De acordo com o TPG e o novo cronograma (matriz de dedicação) refeito em t_l, podemos desenhar o gráfico de Gantt, a partir do qual τ_j^{inicio} e $\tau_j^{fim}(t_l)$ da tarefa τ_j podem ser obtidos.

O custo associado a um cronograma é o total das despesas pagas com empregados disponíveis, de acordo com a dedicação deles às tarefas disponíveis no instante t_l, supondo que não houve variações no esforço das tarefas. Seja t' qualquer tempo durante o desenvolvimento do projeto depois do instante t_l, ou seja, t' representa algum instante entre o fim do projeto e o instante t_l. Seja ainda T_*ativas* o conjunto de tarefas que estão ativas, ou seja, sendo executadas, no momento do tempo t', onde uma tarefa ativa é definida como uma tarefa que não tenha nenhuma tarefa precedente inacabada no TPG em t'. Então, o custo é definido como:

$$f_2(t_l) = custo_I = \sum_{t' \geq t_l} \sum_{e_i \in E\_disp(t_l)} e\_custo_i^{t'}$$

Se $\sum_{j \in \mathrm{T}\_ativas(t')} md_{ij}(t_l) \leq 1$, então

$$e\_custo_j^{t'} = e_i^{salario} \cdot t' \cdot \sum_{j \in \mathrm{T}\_ativas(t')} md_{ij}(t_l)$$

senão, se $1 < \sum_{j \in \mathrm{T}\_ativas(t')} md_{ij}(t_l) \leq e_i^{maxded}$, então

$$e\_custo_i^{t'} = e_i^{salario} \cdot t' \cdot 1 + e_i^{salario\_extra} \cdot t' \cdot \left(\sum_{j \in \mathrm{T}\_ativas(t')} md_{ij}(t_l) - 1 \right)$$

onde $e\_custo_i^{t'}$ significa o valor pago ao empregado e_i no instante t'. Se o total de dedicação de e_i para todas as tarefas ativas ($\sum_{j \in \mathrm{T}\_ativas(t')} md_{ij}(t_l)$) for maior que 1, significa que e_i realiza horas extra em t'.

Para calcular a robustez da solução é usado um método baseado em cenários que consiste em gerar um conjunto $\{\theta_q | q = 1, 2, ..., N\}$ de cenários aleatórios de duração e

custo para as tarefas do projeto. Esses cenários são utilizados no cálculo da robustez do cronograma em um dado instante t_l, dada por:

$$f_3(t_l) = robustez = \sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(\max\left(0, \frac{duracao_q(t_l) - duracao_I}{duracao_I}\right)\right)^2} + \lambda\sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(\max\left(0, \frac{custo_q(t_l) - custo_I(t_l))}{custo_I(t_l)}\right)\right)^2}$$

onde $duracao_I$ e $custo_I$ são respectivamente a duração e o custo inicias do projeto e λ é um parâmetro de peso que determina a importância relativa do custo sobre a duração. Já $duracao_q$ e $custo_q$ são os valores objetivos de eficiência correspondentes em θ_q.

A estabilidade é calculada para todas as tarefas que estão disponíveis no tempo t_l ($t_l > t_0$), e é dada pela soma ponderada das diferenças das dedicações de cada empregado a cada tarefa, em relação ao tempo t_{l-1}. Os pesos (ω_{ij}) servem para penalizar cronogramas nos quais empregados são realocados para uma nova tarefa, simulando o tempo adicional necessário para se familiarizar com ela.

$$f_4(t_l) = estabilidade = \sum_{\{i|e_i \in E\_disp(t_{l-1}) \cap E\_disp(t_l)\}} \sum_{\{j|\tau_j \in \mathrm{T}\_disp(t_{l-1}) \cap \mathrm{T}\_disp(t_l)\}} \omega_{ij}|md_{ij}(t_l) - md_{ij}(t_{l-1})|$$

onde e_i é um empregado, τ_j é uma tarefa, ω_{ij} é um elemento da matriz de pesos, md_{ij} é um elemento da matriz de dedicação, e $E\_disp(t_l)$ e $\mathrm{T}\_disp(t_l)$ são os conjuntos de empregados e tarefas, respectivamente, disponíveis num dado tempo t_l. Os valores dos pesos ω_{ij} são:

$$\omega_{ij} = \begin{cases} 2, & \text{se } md_{ij}(t_{l-1}) = 0 \text{ e } md_{ij}(t_l) > 0, \\ 1,5, & \text{se } md_{ij}(t_{l-1}) > 0 \text{ e } md_{ij}(t_l) = 0, \\ 1, & \text{caso contrário.} \end{cases}$$

No primeiro caso, uma maior penalidade é atribuída se no reescalonamento uma nova tarefa é atribuída a um empregado. No segundo caso, uma penalidade média é atribuída se após o reescalonamento um empregado que estava atribuído a uma tarefa anteriormente e no novo cronograma não está mais alocado para esta tarefa. O último caso ocorre quando um empregado continua trabalhando na tarefa, porém com uma diferença no nível de dedicação, uma pequena penalidade é dada pela diferença entre a nova dedicação e a dedicação original, já que para esses casos o valor do peso é 1.

2.3.7 Restrições

Para que uma solução do DSPSP seja considerada válida, as restrições a seguir devem ser respeitadas:

(i) **Sobrecarga de trabalho**: Em um instante t', após o ponto de reescalonamento t_l, a dedicação total de um empregado disponível para todas as tarefas disponíveis no momento não deve exceder a sua dedicação máxima ao projeto. Por exemplo, caso $e_i^{maxded} = 1,2$, então é permitido ao empregado e_i trabalhar horas extras, desde que não excedam 20% do expediente normal.

$$\forall e_i \in E\_disp(t_l), \forall t' \geq t_l, \text{ onde}$$
$$e\_carga_i^{t'} = \sum_{j \in \mathrm{T}\_ativas(t_l)} md_{ij}(t_l), \text{ e } e\_carga_i^{t'} \leq e_i^{maxded}$$

(ii) **Habilidades da tarefa**: Todos os empregados disponíveis que trabalham juntos em uma tarefa disponível, devem coletivamente abranger todas as habilidades requeridas para esta tarefa. Simultaneamente, cada tarefa disponível em t_l deve ser executada por pelo menos um empregado disponível.

$$\forall j \mid \tau_j \in \mathrm{T}\_disp(t_l)$$
$$req_j \subseteq \bigcup_{e_i \in E\_disp(t_l)} \{hab_i | md_{ij}(t_l) > 0\}$$

(iii) **Limite de empregados por tarefa**: O número de empregados disponíveis trabalhando juntos em uma tarefa τ_j não deve exceder o limite superior $\tau_j^{max\_emp}$. Essa variável é estimada pela fórmula $\tau_j^{max\_emp} = \max\{1, round(2/3(\tau_j^{esf\_tot\_est})^{0.672})\}$, que foi derivada do modelo COCOMO (BOEHM *et al.*, 2000). No ponto de reescalonamento t_l, chamamos de $\tau_j^{tam\_time}(t_l)$ o tamanho da equipe alocada para a tarefa τ_j e de $\tau_j^{num\_min\_emp}(t_l)$ o número mínimo de empregados disponíveis que podem atuar em τ_j para satisfazer essa restrição. Ao contrário das restrições anteriores (*i* e *ii*), esta restrição pode ser relaxada mediante penalização, caso não seja satisfeita.

$$\forall \tau_j \in \mathrm{T}\_disp(t_l), \tau_j^{tam\_time}(t_l) \leq \max(\tau_j^{max\_emp}, \tau_j^{num\_min\_emp}(t_l))$$

2.3.8 Tratamento das restrições

Nos casos em que as soluções encontradas violam alguma restrição, é necessário tratá-las para que sua avaliação não interfira na busca por soluções válidas. Essas restrições são tratadas da seguinte maneira:

(*i*) **Sobrecarga de trabalho**: Quando uma restrição é violada, é comum que o valor da função objetivo sofra uma penalização. No entanto, devido à dificuldade de satisfazer essa restrição, a solução é reparada por meio de normalização para acomodar todas as tarefas nas quais os empregados estão alocados. Em um instante t', caso essa restrição para um empregado e_i seja violada, ou seja, $e\_carga_i^{t'} > e_i^{maxded}$, então sua dedicação $md_{ij}(t_l)$ para cada tarefa τ_j que está sendo executada em t' é dividida por $e\_carga_i^{t'}$. Para possibilitar comparação, o valor normalizado da dedicação $md_{ij}(t_l)$ é denotado por $d_{ij}(t_l)$, de modo que $d_{ij}(t_l) = md_{ij}(t_l)/\max(1, e\_carga_i^{t'}/e_i^{maxded})$. Caso $e\_carga_i^{t'} \leq e_i^{maxded}$, então a dedicação não é normalizada.

(*ii*) **Habilidade da tarefa**: Para incorporar a proficiência de cada empregado para as diferentes tarefas, é preciso saber qual a dedicação total ajustada $A\_Td_j(t_l)$. Então inicialmente obtém a dedicação total $Td_j(t_l)$ de todos os empregados disponíveis para τ_j:

$$Td_j(t_l) = \sum_{e_i \in E\_disp(t_l)} d_{ij}(t_l)$$

Depois, calcula-se o valor total de aptidão (*fitness*) $F_j(t_l)$ de todos os empregados para a tarefa τ_j:

$$F_j(t_l) = \left(\sum_{e_i \in E\_disp(t_l)} e_{ij}^{proficiencia} \cdot d_{ij}(t_l) \right) / Td_j(t_l)$$

onde $F_j(t_l)$ é uma fração da dedicação total gasta pelos empregados na tarefa τ_j, levando-se em consideração a proficiência de cada empregado nas habilidades necessárias para a tarefa. Nos casos em que a proficiência não seja total, o valor da aptidão é reduzido, acarretando em maior tempo para a execução da tarefa. Em seguida, $F_j(t_l)$ é convertido no custo unitário $V_j(t_l)$:

$$V_j(t_l) = \max(1,\ (8 - round(F_j(t_l) * 7 + 0.5)))$$

onde o valor de $V_j(t_l)$ varia no intervalo [1, 7], sendo que quanto menor o valor, mais apto são os empregados alocados para a tarefa τ_j. Após o cálculo de $V_j(t_l)$ é possível obter a dedicação total ajustada $A\_Td_j(t_l)$ de acordo com a aptidão dos empregados:

$$A\_Td_j(t_l) = Td_j(t_l)/V_j(t_l)$$

Conhecida a dedicação total ajustada e sendo $\tau_j^{esf\_rest}(t_l)$ o esforço restante da tarefa τ_j no instante t_l, é possível calcular o tempo requerido para concluir τ_j:

$$\tau_j^{est\_rest}(t_l)/A\_Td_j(t_l) = \tau_j^{est\_rest}(t_l)/(Td_j(t_l)/V_j(t_l))$$

Caso um cronograma candidato em t_l seja inviável porque as habilidades de algumas tarefas não são cobertas pelos empregados disponíveis, então valores elevados de penalidades são atribuídos ao valor das funções objetivo. Sendo $resqsk$ o número de habilidades faltantes, os objetivos são penalizados da seguinte maneira:

$$f_1(t_l) = duracao_I = reqsk \cdot 14k \cdot \sum_{\tau_j \in \mathrm{T}\_disp(t_l)} \tau_j^{est\_tot\_est}(t_l) \Big| \min_{e_i \in E\_disp(t_l)} e_i^{maxded}$$

$$f_2(t_l) = custo_I = reqsk \cdot 14 \cdot \sum_{e_i \in E\_disp(t_l)} \sum_{\tau_j \in \mathrm{T}\_disp(t_l)} e_i^{salario\_extra} \cdot \tau_j^{esf\_tot\_est}(t_l)$$

$$f_3(t_l) = robustez = reqsk \cdot 2 \cdot C_{rob}$$

$$f_4(t_l) = estabilidade = reqsk \cdot 2 \cdot |E\_disp(t_l)| \cdot |\mathrm{T}\_disp(t_l)| \cdot \max_{e_i \in E\_disp(t_l)} e_i^{maxded}$$

onde C_{rob} é uma constante, que neste caso assume o valor de 100 e k é um parâmetro. As penalizações acima garantem que os valores resultantes sejam maiores que os valores dos objetivos de qualquer solução viável válida, conforme demonstrado por Shen *et al.* (2016).

(*iii*) **Limite de empregados por tarefa**: O tratamento dos casos em que o número de empregados alocados a uma tarefa é maior que o limite é feito reparando a solução em duas etapas. Na primeira, atribui-se 0 à dedicação do empregado para a tarefa $md_{ij}(t_l) = 0$, se o mesmo não possuir nenhuma das habilidades requeridas pela tarefa. Na segunda, é verificado se o tamanho da equipe de cada tarefa disponível $\tau_j \in \mathrm{T\_}disp(t_l)$ é maior que o limite de empregados permitido $\tau_j^{max\_emp}$. Se esse limite for excedido $\tau_j^{max\_emp}$, então o seguinte procedimento é executado: 1) ordene os empregados da equipe da tarefa τ_j de acordo com sua proficiência $e_{ij}^{proficiencia}$; 2) iniciando do empregado com a menor proficiência, verifique se pode removê-lo sem violar a restrição (*ii*) e efetue a remoção, caso contrário deixe o empregado na equipe; 3) repita o processo anterior para todos os empregados do time até que todos tenham sido verificados. Caso o tamanho da equipe não possa ser reduzido a $\tau_j^{max\_emp}$ sem violar a restrição (*ii*), então ele pode ser maior que $\tau_j^{max\_emp}$, porém uma penalidade é aplicada ao esforço da tarefa τ_j, como forma de modelar o *overhead* de comunicação necessário para suprir as lacunas de conhecimento da equipe. Sendo assim, o cálculo da penalidade sobre o esforço de τ_j quando $\tau_j^{tam\_time} > \tau_j^{max\_emp}$ é dado por:

$$\tau_j^{esf} = \tau_j^{esf} \cdot \left(1 + \frac{\tau_j^{tam\_time} \cdot (\tau_j^{tam\_time} - 1)/2}{Z}\right)$$

onde τ_j^{esf} é o esforço de τ_j sem considerar o *overhead* e Z é um parâmetro que regula a relação entre o tempo para completar uma tarefa e o número de empregados, que no trabalho da Shen *et al.* (2016) assumiu o valor de 5.

2.3.9 Tomador de decisões

Ao final de cada reescalonamento é preciso simular a tomada de decisão do gerente de projeto sobre a escolha do cronograma a ser adotado para o restante do projeto. O reescalonamento gera um conjunto de soluções não-dominadas, sobre as quais é necessário descobrir se será dada preferência a uma solução que tende a minimizar algum dos objetivos avaliados ou a equilibrá-los. A tomada de decisão consiste nos passos abaixo:

1 Construção de uma matriz par-a-par. A comparação entre os objetivos a serem avaliados visa responder ao questionamento sobre qual deles é mais importante em relação aos demais. Sendo $N\_o = 4$ o número de objetivos, compara-se o objetivo f_i em relação a f_j, sendo $(i, j = 1, 2, ..., N\_o, j > i)$. Então existe um total de $N\_o \cdot (N\_o - 1)/2 = 4(4 - 1)/2 = 6$ comparações para a tomada de decisão. A partir desse ponto pode ser construída uma matriz de comparação $C_1 = (c_{ij})_{N\_o \times N\_o}$ por meio da escala de nove pontos descrita pelo Processo Analítico de Hierarquia (em inglês, *Analitic Hierarchy Process* - AHP) (FÜLÖP, 2005).

2 Estimativa do vetor de peso para múltiplos objetivos. Neste passo, é preciso estimar o vetor de pesos $w = (w_i)_{N\_o \times 1}$ pelo método de mínimos quadrados logarítmico (SAATY; VARGAS, 1984). A média geométrica de cada linha de C_1 é calculada e depois normalizada.

3 Normalização dos valores objetivos. Cada objetivo é normalizado como

$$n\_f_i(\vec{x}) = (f_i^{max} - f_i(\vec{x}))/(f_i^{max} - f_i^{min}), i = 1, 2, ..., N\_o$$

onde f_i^{max} e f_i^{min} são os valores máximo e mínimo dos objetivos entre todas as soluções não-dominadas obtidas no ponto de escalonamento atual.

4 Cálculo do valor de utilidade. A média geométrica ponderada dos múltiplos valores objetivos é usada para encontrar o valor de utilidade de cada solução não-dominada:

$$U(x) = \prod_{i=1}^{N\_o} n\_f_i(\vec{x})^{w_i / \sum_i^{N\_o} w_i}$$

5 Escolha da solução com o maior valor de utilidade. O cálculo do valor de utilidade são baseados nas soluções não-dominadas no ponto do reescalonamento, onde a solução com o maior valor de utilidade será o cronograma vigente.

2.4 Otimização Multiobjetivo

Otimizar significa encontrar uma solução (ou conjunto de soluções) que represente o melhor valor possível para um determinado problema. A medida da qualidade dessa solução é dada por uma ou mais funções objetivo. Em problemas de otimização

com um único objetivo há somente um valor ótimo. Porém, problemas de otimização com mais de uma função objetivo são definidos como problemas de otimização multiobjetivo. Nesses problemas, geralmente as funções estão em conflito e há busca por um balanceamento entre esses objetivos. Assim, o ótimo é definido por um grande conjunto de soluções e então o tomador de decisão escolhe uma solução (ou soluções) que mais se adéquem ao problema (COELLO *et al.*, 2007).

Um problema de otimização é definido da forma descrita a seguir. Seja $\vec{x} \in \mathbb{R}^n$ uma solução para o problema, $f(\vec{x})$ é uma função objetivo que quantifica a adequação de cada solução $\vec{x}$. Além disso, é necessário atender a certas restrições que correspondem a limitações de natureza física ou tecnológica, ou motivos mais subjetivos. Para expressar essas restrições, definem-se regiões no espaço através de desigualdades ou igualdades:

$$g_i(\vec{x}) \leq 0, i = \{1, 2, ..., d\}$$

$$h_j(\vec{x}) = 0, j = \{1, 2, ..., p\}$$

Com essas definições, uma definição formal geral para um problema de otimização consiste em minimizar (ou maximizar) $f(\vec{x})$ sujeita a funções de restrição $g_i(\vec{x}) \leq 0$, $i = \{1, ..., d\}$ e $h_j(\vec{x}) = 0$, $j = \{1, ..., p\}$ $\vec{x} \in \Omega$. Nessa definição, $g_i(\vec{x}) \leq 0$ e $h_j(\vec{x}) = 0$ representam restrições que precisam ser cumpridas ao otimizar $f(\vec{x})$. O termo Ω contém todos os valores possíveis de $\vec{x}$ que podem ser usados para satisfazer uma avaliação de $f(\vec{x})$ e suas restrições.

Um problema de otimização multiobjetivo (em inglês, *Multiobjective Optimization Problem* - MOP) tem como objetivo satisfazer mais de uma função objetivo ao mesmo tempo. Essas otimizações envolvem a maximização (ou minimização) de todas as funções ou uma combinação de maximização e minimização dessas funções. Além disso, essas funções podem mapear objetivos que são conflitantes entre si. Assim, quando o valor de uma função melhora, o de outra piora. O conjunto de soluções de um problema de otimização multiobjetivo é obtido através da Teoria de Otimalidade de Pareto (EHRGOTT, 2005).

Uma solução de problemas de otimização é representada pelo vetor $\vec{x} = (x_1, ..., x_n)$ $\in \Omega$, que é o conjunto de soluções possíveis para o problema. A qualidade da solução é definida por funções das variáveis de decisão, representados por $f_j(\vec{x}) \in \mathbb{R}^n$. Para problemas multiobjetivo, existem várias funções objetivo. Sendo assim, é usado um vetor

de funções objetivo $\overrightarrow{f(\vec{x})} \in \Lambda$, onde Λ é o espaço do vetor das funções objetivo. Um problema multiobjetivo pode ser definido por: $\overrightarrow{f(\vec{x})} = (f_1(\vec{x}), ..., f_m(\vec{x}))$ onde $\overrightarrow{f(\vec{x})}$ pode ser maximizar ou minimizar.

A função da otimização multiobjetivo é otimizar m funções objetivo simultaneamente, de modo a encontrar um conjunto de soluções que representem os melhores resultados entre os objetivos. Alguns conceitos fundamentais as técnicas de otimização multiobjetivo, definidas por (COELLO *et al.*, 2007), são:

Dominância de Pareto: É dito que um vetor de funções objetivo $\overrightarrow{f(\vec{x})} = (f_1(\vec{x}), ..., f_m(\vec{x}))$ *domina* outro vetor $\overrightarrow{f(\vec{y})} = (f_1(\vec{y}), ..., f_m(\vec{y}))$, denotado por $\overrightarrow{f(\vec{x})} \preceq \overrightarrow{f(\vec{y})}$, se e somente se $\overrightarrow{f(\vec{x})}$ é parcialmente menor que $\overrightarrow{f(\vec{y})}$, em outras palavras, $\forall_i \in \{1, ..., m\}, f_i(\vec{x}) \leq f_i(\vec{y}) \wedge \exists_i \in \{1, ..., m\} : f_i(\vec{x}) < f_i(\vec{y})$.

Otimalidade de Pareto: Uma solução $\vec{x} \in \Omega$ é dita *ótimo de Pareto* quando não existe um $\vec{y} \in \Omega$ para qual $f(\vec{y}) = (f_1(\vec{y}), ..., f_m(\vec{y}))$ domina $f(\vec{x}) = (f_1(\vec{x}), ..., f_m(\vec{x}))$.

Conjunto Ótimo de Pareto: Para um MOP, há mais de uma solução ótima de Pareto. Sendo assim, o *conjunto ótimo de Pareto* P^* é definido por:

$$P^* : = \{\vec{x} \in \Omega | \neg \exists \vec{x'} \in \Omega | f(\vec{x'}) \preceq f(\vec{x})\}$$

Fronteira de Pareto: Dado um MOP e um conjunto ótimo de Pareto (P^*), a *fronteira de Pareto* (PF^*) é definida como:

$$PF^* = \{\vec{z} = f(\vec{x}) | \vec{x} \in P^*\}$$

Conhecida a fronteira de Pareto, o tomador de decisão pode escolher as soluções desejadas de acordo com o desempenho obtido para cada objetivo.

2.4.1 Otimização com Muitos Objetivos

De modo geral, quando o número de objetivos cresce, os algoritmos têm maior dificuldade em se aproximar do ótimo. Problemas de otimização multiobjetivo com quatro ou mais funções objetivo são chamados de problemas de otimização com muitos objetivos. Apesar de Nakayama *et al.* (1995) afirmar que um limite superior no número de objetivos para problemas de otimização com muitos objetivos não seja tão claro, exceto em algumas ocasiões, a maioria dos profissionais está interessado em um máximo de 10 à 15 objetivos (DEB; JAIN, 2014).

Os MOEAs quando aplicados a problemas de otimização com muitos objetivos têm uma escalabilidade ruim (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008). Então, os pesquisadores têm investigado a escalabilidade desses algoritmos em relação ao número de funções objetivo. As principais dificuldades encontradas pelos MOEAs nesse tipo de problema são:

- **Deterioração da habilidade de busca**: Com o aumento do número de objetivos, a quantidade de soluções não-dominadas também aumenta (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008). Como resultado disso, a habilidade de busca dos algoritmos que usam a dominância de Pareto é deteriorada.
- **Dimensão da fronteira de Pareto**: Esse problema é dado pela quantidade de soluções para se aproximar da fronteira de Pareto. Nesse caso, o conjunto ótimo de Pareto cresce de maneira excessiva à medida que o número de funções objetivo aumenta.
- **Visualização da fronteira de Pareto**: A visualização dos pontos da fronteira de Pareto é importante para a tomada de decisão (ISHIBUCHI; TSUKAMOTO; NOJIMA, 2008). Porém, com mais de três dimensões não é possível plotar os pontos da fronteira de Pareto de maneira convencional.

Schutze, Lara e Coello (2011) destacam que os desafios que os MOEAs devem enfrentar em problemas com muitos objetivos são: manter um bom conjunto de soluções, com o intuito de direcionar a população em direção ao conjunto de interesse; e aumentar a probabilidade de melhorar um indivíduo da população. Para a diminuição desses

problemas, algumas propostas têm sido apresentadas e categorizadas em (ISHIBUCHI *et al.*, 2011):

- **Adaptação das relações de preferência**: Essa abordagem tenta aumentar a pressão em direção da fronteira de Pareto através de relações de preferência para induzirem uma nova ordenação do espaço de objetivos (JAIMES; QUINTERO; COELLO, 2009);
- **Redução da dimensionalidade**: Nessa abordagem, a ideia é identificar os objetivos menos conflitantes e descartá-los, sem que haja uma alteração no conjunto ótimo de Pareto;
- **Estratégias de decomposição**: Essa abordagem usa métodos de decomposição, onde os métodos decompõem o problema multiobjetivo em vários problemas escalares e o EA é aplicado para otimizar esses subproblemas (ZHANG; LI, 2007);
- **Incorporação de preferências**: Nessa abordagem, um conjunto de métodos usa informações de preferências, dada pelo tomador de decisão, para executar um algoritmo para muitos objetivos que se concentra em determinada região da fronteira de Pareto (PURSHOUSE; JALBĂ; FLEMING, 2011);
- **Diferentes esquemas de avaliação de *fitness***: Essa abordagem não usa a dominância de Pareto como função de *fitness*, mas um conjunto diferente de medidas de avaliação, como por exemplo um conjunto de indicadores, tais como o hipervolume e epsilon (ZITZLER; KÜNZLI, 2004).

Neste trabalho, o algoritmo utilizado incorpora a estratégia de decomposição para lidar com problema com muitos objetivos.

2.4.2 Otimização Dinâmica

Boa parte dos problemas do mundo real são compostos por vários objetivos concorrentes e que possuem características que mudam ao decorrer do tempo. Adicionado a isso, ainda existe a incerteza inerente a variáveis, restrições e até aos objetivos. O desenvolvimento de estratégias que trabalham com cenários dinâmicos e imprecisos

são um desafio para a comunidade científica da área. Muitos problemas nesse contexto podem ser modelados como problemas de otimização dinâmica (em inglês, *Dynamic Optimization Problems* - DOPs) (CRUZ; GONZÁLEZ; PELTA, 2011).

O dinamismo dessa classe de problemas é dado pela ocorrência de eventos que modificam suas características e por ser dependente do tempo. Exemplos de eventos podem ser o aumento/diminuição de variáveis de decisão, modificações nas restrições, dentre outros (NGUYEN; YANG; BRANKE, 2012). No contexto do DSPSP, por exemplo, os eventos que podem afetar as instâncias de um problema são a chegada de novas tarefas, demissão de empregados, mudanças na disponibilidade de recursos, entre outros.

Segundo Cruz *et al* (2011), um DOP (*Dynamic Optimization Problem*) é aquele que busca otimizar uma função $f(\vec{x}, t)$ sujeita a restrição $\vec{x} \in F(t)$ e $F(t) \subseteq S, t \in T$, onde:

- $S \in \mathbb{R}^n$, sendo S o espaço de busca.
- t é o tempo.
- $f: S \times T \rightarrow \mathbb{R}$, é a função objetivo que atribui um valor numérico ($f(\vec{x}, t) \in \mathbb{R}$) para cada possível solução ($\vec{x} \in S$) no tempo t.
- $F(t)$ é o conjunto de soluções viáveis, $F(t) \subseteq S$, no tempo t.

Em outras palavras, um problema de otimização dinâmica é um problema onde a função objetivo ou as restrições mudam com o tempo. Sendo assim, o objetivo de métodos que lidam com DOPs não é mais localizar uma solução ótima estacionária, mas rastrear seu movimento pelos espaços de soluções e tempo, tanto quanto possível.

Na otimização dinâmica, a suposição geral é que o problema depois de uma mudança está de alguma maneira relacionada com o problema antes da mudança. Então, um algoritmo de otimização precisa aprender com suas experiências de buscas passadas, com a finalidade de avançar na busca mais efetivamente. Nguyen, Yang e Branke (2012) sumarizam algumas abordagens desenvolvidas por pesquisadores que lidam com DOPs, são elas:

- **Detecção de mudanças**: Muitas abordagens de otimização dinâmica evolucionária tomam ações explícitas para responder a mudanças no ambiente. Essas mudanças são detectadas pela reavaliação de soluções ou pelo comportamento do algoritmo.

- **Introdução de diversidade quando mudanças ocorrem**: Na otimização dinâmica a convergência do algoritmo não é necessariamente algo desejável. Isso se dá porque as mudanças no espaço de soluções ocorrem em uma área e não existe nenhum membro do algoritmo nessa área, então o algoritmo falha em detectar o ótimo global móvel.

- **Manutenção da diversidade durante a busca**: Uma outra abordagem é manter a diversidade da população ao longo da busca, para evitar a possibilidade que toda a população convirja para um único lugar. Esse método não precisa detectar as mudanças explicitamente, pois baseia-se na diversidade para adaptar-se a elas.

- **Abordagens de memória**: Nos casos em que as mudanças são periódicas ou recorrentes, pode ser útil reutilizar soluções encontradas previamente para economizar tempo computacional e para influenciar o processo de busca.

- **Abordagens preditivas**: Existem casos em que as mudanças nos ambientes dinâmicos podem apresentar alguns padrões que são previsíveis. Para esses casos, pode ser interessante aprender os tipos de padrões a partir de experiências de buscas passadas, com a finalidade de tentar prever mudanças no futuro.

- **Métodos auto adaptativos**: É possível lidar com as mudanças no ambiente de busca com mecanismos auto adaptativos dos algoritmos evolucionários. Esses mecanismos, em geral, permitem evoluir os parâmetros da estratégia de mutação do algoritmo, baseado na aptidão da população.

- **Abordagens multipopulação**: Essa abordagem pode ser vista como uma combinação das abordagens de manutenção de diversidade, memória e auto adaptação. Essa abordagem mantém múltiplas subpopulações concorrentemente, onde cada subpopulação lida com uma área separada do espaço de busca e pode ter responsabilidade para diferentes tarefas. Por exemplo, algumas subpopulações focam em procurar o global ótimo enquanto outras subpopulações podem focar em rastrear as possíveis mudanças.

As abordagens de otimização dinâmica utilizadas nesse trabalho foram a detecção de mudança, abordagens de memória, métodos auto adaptativo e multipopulação.

2.5 Algoritmos Meta-heurísticos

Para a resolução de problemas multiobjetivo, geralmente são utilizadas meta-heurísticas populacionais. A característica paralela das meta-heurísticas populacionais é um mecanismo utilizado para controlar a exploração e a intensificação da busca, gerando assim, em uma só execução do algoritmo, uma aproximação da fronteira de Pareto, buscando ao mesmo tempo diversidade e convergência. Nesta seção detalharemos os dois algoritmos abordados no trabalho: Evolução diferencial e NSGA-III.

2.5.1 Evolução Diferencial (DE)

O algoritmo de DE foi desenvolvido por Storn e Price em 1995 (PRICE; STORN, 1995). Ele tem uma estrutura semelhante a do EA, mas se difere dos tradicionais EAs na sua geração de novas soluções candidatas e pelo uso de um esquema de seleção "gulosa". O DE é uma versão melhorada do GA para otimizações mais rápidas (BABU; JEHAN, 2003). No DE, cada valor das variáveis no cromossomo é representado por um valor de número real. As vantagens do DE são sua estrutura simples, fácil de usar, velocidade e robustez (DAS; ABRAHAM; KONAR, 2008).

O algoritmo funciona criando uma população inicial aleatória de potenciais soluções, garantindo-se, pela aplicação de regras que o valor de cada variável está dentro de seus limites. Um indivíduo é selecionado aleatoriamente para ser substituído e três diferentes indivíduos são selecionados como pais. Um desses três indivíduos é escolhido como o pai principal. Cada variável do vetor pai principal tem uma probabilidade de ser alterada. A mudança é realizada adicionando ao valor da variável uma relação da diferença entre os dois valores dessas variável nos outros dois pais. Em resumo, o vetor do pai principal é perturbado com o vetor dos outros dois pais. Esse processo representa a operação de *crossover* no DE. Se esse vetor resultante for melhor do que o escolhido para ser substituído ele é trocado, caso contrário ele é mantido e passa para a próxima iteração do algoritmo.

O esquema adaptativo usado pelo DE garante que suas mutações sejam automaticamente escaláveis, além de usar um *crossover* não uniforme, ele também usa a seleção via torneio, onde o filho compete com um de seus pais.

2.5.2 Cooperative Multiobjective Differential Evolution (CMODE)

O *Cooperative MultiObjective Differential Evolution* (CMODE) é um algoritmo de otimização baseado em cooperação com múltiplas populações. O algoritmo foi proposto por Wang, Zhang e Zhang (2016), e ao invés de lidar com todos os objetivos de uma só vez, considerando a mesma população como em abordagens tradicionais, o CMODE usa múltiplas populações para lidar com múltiplos objetivos. Cada população lida com apenas um objetivo e todas as populações cooperam para aproximar toda a PF^*. Em resumo, o CMODE é caracterizado pelo uso de múltiplas populações para resolver problemas do tipo MOP por meio de cooperação.

O CMODE tem várias vantagens, entre elas, ele é muito simples e utiliza DEs de objetivo único avançados com parâmetros adaptáveis para a otimização multiobjetivo. Os DEs cooperam em dois níveis: (*i*) um entre subpopulações mono-objetivo e (*ii*) outro entre subpopulações e o arquivo populacional. O CMODE também pode ser considerado como um MOEA paralelo, usando o modelo mestre-escravo, onde as subpopulações mono-objetivo são processadores escravos (trabalhadores) e o arquivo populacional é o processador central. Além disso, o CMODE pode ser aplicado para resolver não apenas problemas com dois ou três objetivos, mas também problemas com muitos objetivos.

O CMODE possui M subpopulações de otimização mono-objetivo e um arquivo de população para problema de otimização de M-objetivos. Cada subpopulação mono-objetivo é usada para otimizar um objetivo correspondente do MOP. O arquivo população é usado não apenas para manter todas as soluções não-dominadas encontradas por todas as subpopulações até o momento, mas também para orientar a busca de cada subpopulação mono-objetivo ao longo de toda a fronteira de Pareto. Essas $(M+1)$ populações cooperam para otimizar todos os objetivos do MOP.

Mais especificamente, em cada geração (G), os valores de aptidão de um indivíduo na M-subpopulação são atribuídos pela M-ésima função objetivo do MOP. Assim, os indivíduos não são confundidos por diferentes objetivos conflitantes e poderiam ser

guiados pelo objetivo correspondente para buscar diferentes regiões da PF^*. No entanto, como cada subpopulação se concentra na otimização de apenas um objetivo, os indivíduos em cada subpopulação podem ser guiados para a margem ou ponto extremo do objetivo correspondente, resultando em uma aproximação ineficiente de toda a PF^*. Para remediar esse problema, diferentes subpopulações mono-objetivo compartilham suas informações de busca e se comunicam com outras através do arquivo de população para aproximar toda a PF^* até certo ponto. Além disso, o arquivo populacional também pode ser otimizado para aproximar toda a PF^*. Assim, cada subpopulação para busca local e o arquivo populacional para busca global coopera para aproximar toda a PF^*.

À primeira vista, o CMODE parece ser mais complicado do que uma abordagem geral baseada em Pareto, já que gerencia múltiplas populações. Na verdade, o CMODE é bem simples. As vantagens do CMODE em relação às abordagens tradicionais são as seguintes:

- No arquivo populacional, a busca é implementada diretamente em todas as soluções não-dominadas encontradas. Então, apenas o operador de comparação da dominância de Pareto é usado para coletar soluções não-dominadas. Não é necessário adotar nenhum esquema de atribuição de *fitness*.
- Em cada subpopulação, o processo de otimização mono-objetivo é seguido e implementado diretamente para cada objetivo, portanto os indivíduos não são mais confundidos por diferentes objetivos conflitantes. Também não é necessário adotar um esquema elaborado de atribuição de *fitness* para classificar e selecionar bons indivíduos.
- Cada subpopulação para busca local e o arquivo populacional para busca global cooperam para aproximar toda a fronteira de Pareto.

O CMODE é um algoritmo baseado em múltiplas populações, com um aspecto único no projeto do algoritmo: (*i*) o DE adaptativo é usado para cada subpopulação mono-objetivo; (*ii*) o DE adaptativo também é usado para a evolução do arquivo; e (*iii*) uma estratégia de atualização do arquivo avançada é introduzida para resolver problemas com muitos objetivos.

Os três componentes principais, como o DE para subpopulações mono-objetivo, o DE para o arquivo populacional e o esquema de atualização do arquivo estão sublinhados na Figura 2.6, que mostra a estrutura da população do CMODE e esses principais componentes.

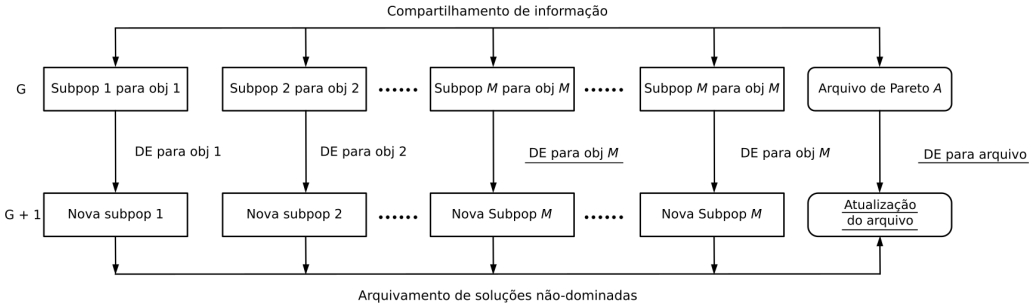

Figura 2.6 – CMODE: estrutura da população e principais componentes (adaptado de (WANG; ZHANG; ZHANG, 2016))

O procedimento completo do CMODE é descrito no Algoritmo 1. Na inicialização (linhas 1 a 11), o arquivo A é marcado como vazio. Então M subpopulações com NP indivíduos em cada subpopulação são inicializadas aleatoriamente. Para finalizar, o arquivo populacional A é atualizado extraindo as soluções não-dominadas de todas as subpopulações.

Em cada geração (linhas 13 a 33), cada indivíduo é atualizado através da execução do DE para as subpopulações. Nas linhas 16 e 17, os parâmetros adaptativos de mutação e *crossover* do indivíduo X_i^m são gerados. O fator de mutação F_i^m, para cada geração G, é gerado independentemente de acordo com uma distribuição Cauchy com o parâmetro de localização μ_F^m e o parâmetro de escala 0,1. Similarmente, para cada geração G, as probabilidades do *crossover* CR_i^m são geradas independentemente de acordo com uma distribuição normal de média μ_{CR}^m e desvio padrão 0,1. Na linha 19, é gerado um vetor mutante V_i^m, onde $A_{r,G}$ é uma solução não-dominada do arquivo compartilhado para X_i^m. A diferença adicional do termo $F_i^m \cdot (A_{r,G} - X_{i,G}^m)$ é usado para compartilhar informações do arquivo, portanto o indivíduo X_i^m pode ser usado para informações de busca não apenas da sua própria subpopulação, mas também de outras populações. Na linha 20, o operador de *crossover* é aplicado para cada par do vetor alvo $X_{i,j,G}$ e o vetor

mutante V_i^m para gerar um vetor experimental U_i^m. Esse vetor é gerado através de um *crossover* binomial, onde $rand_j(0,1)$ é um número aleatório uniforme em [0,1] para j-ésima dimensão, $CR \in [0,1]$ é um parâmetro de controle do *crossover* predefinido, e $j_{rand} \in [1, D]$ é um inteiro aleatoriamente escolhido de 1 até a dimensão D. Nas linhas 24 a 29, é escolhido o indivíduo com o menor valor do objetivo (para problemas de minimização). Na linha 30, é atualizado o parâmetro de localização μ_F^m, onde S_F^m denota o conjunto de todos os fatores de mutação bem sucedidos na geração G, e $mean_L(\cdot)$ representa a média de Lehmer. Na linha 31, é atualizado a média μ_{CR}^m, onde S_{CR}^m denota o conjunto de todas as probabilidades de *crossover* bem sucedidos, c é uma constante de valor positivo entre 0 e 1 (nesse caso, $c = 0.1$), e $mean_A(\cdot)$ é a média aritmética.

Após todas as subpopulações serem atualizadas, o arquivo populacional A é evoluído de acordo com o Algoritmo 2 e então é atualizado de acordo com o esquema de atualização do arquivo. Esse processo se repete até que o critério de aceitação seja finalizado, fazendo com que as soluções do arquivo A sejam exibidas. No esquema de atualização do arquivo (linhas 35 a 38), primeiro, todas as soluções geradas pelas M subpopulações, a população atual do arquivo A, e sua população de descendentes A' são coletadas e combinadas. Então, todas as soluções não-dominadas são extraídas das soluções combinadas. Todas as soluções não-dominadas obtidas são definidas como os novos membros do arquivo. Após essa etapa, é preciso verificar se o tamanho do arquivo excede o tamanho limite predefinido, já que o número de soluções não-dominadas pode ser enorme e a complexidade computacional de mantê-las é alta. O processo de truncamento do arquivo é para selecionar soluções menos congestionadas e descartar membros do arquivo que são densamente distribuídos com base na estimativa de densidade.

O procedimento do DE para o arquivo populacional é descrito no Algoritmo 2. Nas linhas 6 e 7, o fator de mutação F_i e o *crossover* CR_i para cada indivíduo é definido, onde $rand[0,1]$ é um número aleatório uniformemente distribuído entre 0 e 1, e $\rho_1 = \rho_2 = 0.1$ indica probabilidades para ajustar F_i e CR_i. Na linha 8 é gerado o vetor mutante V_i, e na linha 9 o vetor experimental U_i é gerado igual a linha 20 do Algoritmo 1.

2.5.2.1 Extensão do CMODE para otimização com muitos objetivos

A maioria dos MOEAs existentes são projetados e testados em problemas com dois ou três objetivos (ZHOU *et al.*, 2011). A performance desses algoritmos geralmente

Algoritmo 1 CMODE

arquivo $A = \emptyset$; geração $G = 0$; número de avaliações da função objetivo $NFE = 0$; $c = 0.1$;
/*inicialização*/
para cada subpopulação $m = 1$ até M **faça**
  $\mu_{CR}^m = 0.5; \mu_F^m = 0.5$;
  para cada indivíduo $i = 1$ até NP na m-ésima subpopulação **faça**
    inicialize aleatoriamente o vetor alvo X_i^m;
    avalie X_i^m;
    $NFE = NFE + 1$;
  fim para
  atualize $X_{melhor}^m = \arg\min\{f_m(X_i^m) | i = 1, 2, ..., NP\}$;
fim para
atualiza o arquivo populacional A extraindo as soluções não-dominadas a partir de todas as subpopulações; $F_i = 0.5$ e $CR_i = 0.9$ para cada membro do arquivo;
enquanto o critério de parada não for aceito **faça**
  /*DE para subpopulações*/
  para cada subpopulação $m = 1$ até M **faça**
    $S_F^m = \emptyset; S_{CR}^m = \emptyset$;
    para cada indivíduo $i = 1$ até NP na m-ésima subpopulação **faça**
      $F_i^m = randc_i(\mu_F^m, 0.1)$;
      $CR_i^m = randn_i(\mu_{CR}^m, 0.1)$;
      selecionar aleatoriamente um indivíduo A_r do arquivo A;
      $V_i^m = X_{i,G}^m + F_i^m \cdot (X_{melhor,G}^m - X_{i,G}^m) + F_i^m \cdot (X_{i_1,G}^m - X_{i_2,G}^m) + F_i^m \cdot (A_{r,G} - X_{i,G}^m)$
      crie o vetor U_i^m através do *crossover* entre os vetores $X_{i,j,G}$ e $V_{i,j,G}$, sendo
      que o cromossomo $u_{i,j,G} = \begin{cases} v_{i,j,G}, & \text{se } rand_j(0,1) \leq CR \text{ ou } j = j_{rand} \\ x_{i,j,G}, & \text{caso contrário} \end{cases}$
      avalie U_i^m;
      $NFE = NFE + 1$;
    fim para
    para cada indivíduo $i = 1$ até NP na m-ésima subpopulação **faça**
      se $f_m(U_i^m) < f_m(X_i^m)$ **então**
        $X_i^m = U_i^m$;
        $CR_i^m = S_{CR}^m; F_i^m = S_F^m$;
      fim se
    fim para
    $\mu_F^m = (1 - c) \cdot \mu_F^m + c \cdot mean_L(S_F^m)$
    $\mu_{CR}^m = (1 - c) \cdot \mu_{CR}^m + c \cdot mean_A(S_{CR}^m)$
    $X_{melhor}^m = \arg\ \min\{f_m(X_i^m) | i = 1, 2, ..., NP\}$;
  fim para
  /*Algoritmo 2*/
  DE para o arquivo populacional A para geração da população de descendentes A';
  /*Esquema de atualização do arquivo*/
  $A = SolucoesNaoDominadas$(todas as subpopulações $\cup\ A \cup A'$);
  se ($tamanho(A) >$ tamanho limite predefinido) **então**
    truncamento do arquivo baseado na estimativa de densidade;
  fim se
  $G = G + 1$;
fim enquanto
retorne A;

Algoritmo 2 DE para o arquivo populacional

1: $A' = \emptyset; AS = tamanho(A)$;
2: **se** $(AS < 4)$ **então**
3: retorne;
4: **fim se**
5: **para** cada indivíduo $i = 1$ até AS no arquivo populacional **faça**
6: $F_i = \begin{cases} rand_i[0.1, 1], & rand[0, 1] < \rho_1 \\ F_i, & \text{caso contrário} \end{cases}$
7: $CR_i = \begin{cases} rand_i[0, 1], & rand[0, 1] < \rho_2 \\ CR_i, & \text{caso contrário} \end{cases}$
8: $V_i = X_{i_1} + F_i \cdot (X_{i_2} - X_{i_3})$;
9: gere U_i igual a linha 20 do Algoritmo 1;
10: avalie U_i;
11: $NFE = NFE + 1$;
12: $A' = A' \cup U_i$;
13: **fim para**
14: retorne A';

se deteriora à medida que o número de objetivos aumenta. A maioria dos algoritmos clássicos baseados em Pareto, como NSGA-II e SPEA-II, não podem fornecer pressão de seleção suficiente na direção da fronteira de Pareto, para a maioria dos problemas com muitos objetivos. Uma das principais razões é que a proporção de soluções não-dominadas em uma população tende a aumentar, à medida que o número de objetivos aumenta. Isso faz com que o critério de seleção primário, baseado na relação de dominância de Pareto, falhe em distinguir soluções e o segundo critério de seleção, baseado na densidade, desempenhe um papel de liderança, tanto na seleção de acasalamento quanto na seleção ambiental de um algoritmo (DEB; JAIN, 2014). Apesar do trabalho de Wang, Zhang e Zhang (2016) não ter como principal propósito tratar problemas com muitos objetivos, eles tentaram estender o CMODE para lidar com este fato. Para isso, os autores introduziram no CMODE a estratégia *shift-based density estimation* (SDE) para o truncamento do arquivo. Essa extensão do CMODE para muitos objetivos foi chamada de CMODE+SDE.

Em geral, a técnica de estimativa de densidade estima a densidade de um indivíduo considerando a relação de posição mútua entre ele e outros indivíduos na população. Ao estimar a densidade de um indivíduo X, o SDE desloca as posições de outros indiví-

duos na população de acordo com a comparação da convergência entre esses indivíduos e X, em cada objetivo. Especificamente, se um indivíduo tiver um desempenho melhor que X para um objetivo, ele será deslocado para a mesma posição de X nesse objetivo; caso contrário, permanece inalterado.

A ideia principal do SDE é atribuir valores de alta densidade às soluções pouco convergentes, colocando-as em áreas lotadas. Então, essas soluções pouco convergentes podem ser filtradas pelo segundo critério de seleção. O SDE pode ser aplicado para o estimador de densidade do CMODE apenas usando a versão deslocada de indivíduos para calcular a distância. A implementação do SDE é simples, com custo computacional insignificante e sem a necessidade de parâmetros adicionais.

2.5.3 Non-dominated Sorting Genetic Algorithm III (NSGA-III)

O NSGA-III é um algoritmo com muitos objetivos proposto por Deb *et al* (DEB; JAIN, 2014). O *framework* básico permanece similar ao original NSGA-II (DEB *et al.*, 2002) com significantes mudanças no mecanismo de seleção. Para isso, primeiro a população pai P_g de tamanho N é aleatoriamente inicializada, e então um torneio de seleção binária, *crossover* e mutação são aplicados para criar uma população filha Q_g. Depois disso, ambas populações são combinadas e ordenadas de acordo com seu nível de dominância e os melhores N membros são selecionados da população combinada da população pai e da próxima geração. A diferença fundamental entre o NSGA-II e o NSGA-III está na forma como a operação de preservação do nicho é realizada.

Diferente do NSGA-II, o NSGA-III começa com um conjunto de pontos de referência PR^r. Depois da ordenação não-dominada, todos os membros da fronteira aceitos e a última fronteira F_l que não puderam ser completamente aceitos são salvos em um conjunto S_g. Membros em S_g/F_l são selecionados para a próxima geração. Entretanto, os membros restantes são selecionados de F_l para que uma diversidade seja mantida na população. No NSGA-III, os pontos de referência fornecidos (PR^r) são usados para selecionar esses membros remanescentes (Figura 2.7). Para realizar isso, os valores objetivo e os pontos de referência são normalizados para que eles tenham uma faixa idêntica.

A manutenção da diversidade entre os membros da população no NSGA-III é

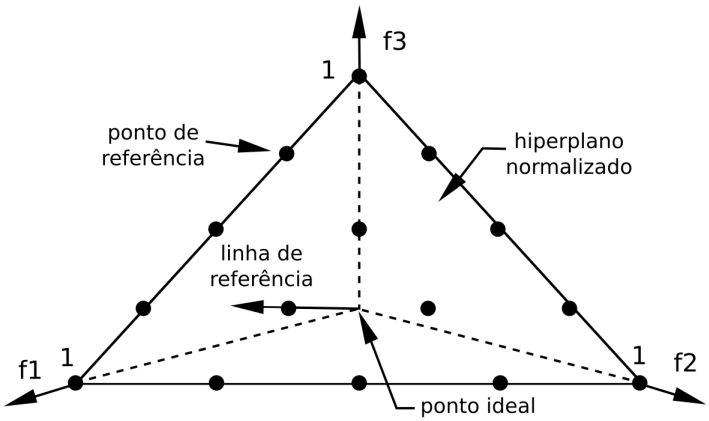

Figura 2.7 – Plano de referência normalizado para três objetivos (adaptado de (DEB; JAIN, 2014))

auxiliada pelo fornecimento e atualização adaptativa de uma série de pontos de referência bem disseminados. O pseudo-código do procedimento do NSGA-III para a geração g é mostrado no Algoritmo 3.

Algoritmo 3 NSGA-III

Entrada: H pontos de referência estruturados PR^s, população pai P_g
Saída: P_{g+1}

$S_g \leftarrow \emptyset, i \leftarrow 1$
$Q_g \leftarrow Recombinação + Mutação(P_g)$
$R_g \leftarrow P_g \cup Q_g$
$(F_1, F_2, ...) \leftarrow Ordenação\ não - dominada(R_g)$
repita
    $S_g \leftarrow S_g \cup F_i$
    $i \leftarrow i + 1$
até $|S_g| \geq N$
$F_l \leftarrow F_i$ /* última fronteira a ser incluída */
se $|S_g| = N$ **então**
    $P_{g+1} \leftarrow S_g$
senão
    $P_{g+1} \leftarrow \cup_{j=1}^{l-1} F_j$
    /*números de pontos para ser escolhidos de F_l*/
    $K \leftarrow N - |P_{g+1}|$
    /*Normaliza os objetivos e cria o conjunto de referência PR^{r*}*/
    $Normalizar(F^M, S_g, PR^r, PR^s)$
    /*Associa cada membro s de $S + g$ com um ponto de referência*/
    /*$\pi(s)$: ponto de referência mais próximo*/
    /*$d(s)$: distância entre s e $\pi(s)$*/
    $[\pi(s), d(s)] \leftarrow Associar(S_g, PR^r)$
    /*Computa a contagem de nicho do ponto de referência $j \in PR^r$*/
    $\rho_j \leftarrow \sum_{s \in S_g/F_l}((\pi(s) = j)?1 : 0)$
    /*Escolhe K membros, um por vez, de F_l para construir P_{g+1}*/
    $Niching(K, \rho_j, \pi, d, PR^r, F_l, P_{g+1})$
fim se

2.6 Trabalhos Relacionados

Como parte deste trabalho, foi conduzida uma revisão sistemática (REZENDE *et al.*, 2019) onde foram identificados os principais trabalhos relacionados ao SPSP no contexto da SBSE, entre outros elementos. A seguir são apresentados alguns dos principais trabalhos da área encontrados na revisão.

O uso de algoritmos evolucionários aplicados a problemas da área de gerencia-

mento de projetos foi inicialmente explorado por (CHANG *et al.*, 1998) introduzindo o *Software Project Management net* (SPM-Net), dando início a exploração do uso de algoritmo genético para alocação e escalonamento de recursos. Nos trabalhos de (ANTONIOL; PENTA; HARMAN, 2004b) e (ANTONIOL; PENTA; HARMAN, 2005) foram utilizadas algoritmos de HC, GA e *Simulated Annealing* (SA) para o problema de alocação de recursos em pacotes de trabalhos com o objetivo específico de reduzir a duração do projeto.

Alba e Chicano (2005, 2007) também aplicaram algoritmos de busca em projetos de software. No trabalho de 2007 eles definiram uma das formulações mais populares para o problema de escalonamento de projetos de software, o SPSP. O trabalho apresenta uma abordagem que utiliza GA para encontrar uma solução ótima. Nesses trabalhos procuraram minimizar o tempo e custo do projeto, tendo como entrada informações sobre os membros da equipe, as tarefas e a relação de dependência entre elas. Eles combinaram os objetivos de tempo e custo em uma única função objetivo que é constituída pela soma ponderada simples para calcular a qualidade de uma solução. Essa abordagem foi avaliada utilizando um conjunto de instâncias sintéticas, geradas por um gerador de instâncias.

Posteriormente, Chicano *et al.* (2011) e Luna *et al.* (2011) estenderam esse trabalho para lidar com múltiplos objetivos usando a abordagem de otimização de Pareto, utilizando alguns algoritmos clássicos de otimização multiobjetivo como NSGA-II, SPEA2, PAES, MOCell, GDE3, MO-FA e DEPT.

Alguns trabalhos investigaram as limitações da formulação do SPSP. Antoniol *et al* (2004a) observaram que os parâmetros são baseados em estimativas que podem estar erradas ou com um elevado grau de incerteza, devido a fatores como retrabalho, ou até mesmo abandono de pacotes de trabalho, por diversos motivos. Usando dados de um projeto real, Antoniol *et al* (2005) estudaram técnicas baseadas em busca para projetos de manutenção de grande porte, que tem como característica a rigidez e alto risco de afetar sistemas em produção na organização.

Outras abordagens e formulações foram introduzidas para o problema do planejamento do projeto de software. Alvarez-Valdes *et al.* (2006) desenvolveram um algoritmo baseado em *Scatter Search* para problemas de escalonamento de projetos baseados em

recursos parcialmente renováveis. Esse tipo de recurso não é consumido ao longo do projeto, porém só se encontra disponível por um período limitado. Hericko *et al* (2008) procuraram descobrir como otimizar o tamanho da equipe de projeto em função do tamanho do software e do esforço, usando algoritmo de otimização baseado em gradiente. Kang *et al* (2011) otimizaram a alocação de recursos humanos por meio do uso de uma variante do SA, o ASA (*Accelerated Simulated Annealing*), considerando uma série de restrições que, quando violadas impõem penalidades às soluções. Chen e Zhang (2013) usaram a técnica de ACO (*Ant Colony Optimization*), em conjunto com um escalonador baseado em eventos (em inglês, *Event-Based Scheduler* - EBS), para ajustar a alocação de empregados na ocorrência de eventos de adição ou remoção de empregados. Luna *et al.* (2014) abordaram a escalabilidade das técnicas de otimização multiobjetivo relacionados ao crescimento no tamanho dos projetos de software, implicando em espaços de busca cada vez maiores. Wu *et al.* (2016) propuseram a abordagem de uma hiper-heurística evolutiva para resolver o SPSP, usando dois algoritmos genéticos para isso. Essa abordagem permite buscar por uma melhor heurística ao invés de uma melhor solução, permitindo escolher tanto o operador de mutação quanto o de cruzamento. Shen *et al.* (2016) propuseram uma extensão do modelo de Alba e Chicano (2007) para um modelo dinâmico e um algoritmo evolutivo adaptado para esse modelo. Recentemente, Shen *et al.* (2018) propuseram um novo algoritmo para o modelo dinâmico chamado MOTAMAQ.

Poucos trabalhos abordam o DSPSP, e nenhum deles usam a estrutura do algoritmo de evolução diferencial nessa abordagem. Além disso, só o trabalho de (SHEN *et al.*, 2016; SHEN *et al.*, 2018) usa algumas técnicas de otimização dinâmica.

Em outros contextos, Mendes e Mohais (2005) propuseram uma abordagem usando o DE para resolver problemas de otimização dinâmica, chamado DynDE, um algoritmo DE multipopulação desenvolvido especificamente para otimizar funções objetivo que variam lentamente no tempo. O DynDE não precisa de nenhuma estratégia de controle de parâmetro para mutação ou *crossover* (F ou CR), onde os resultados obtidos mostraram que essa abordagem é competitiva em relação às outras abordagens existentes na área de otimização dinâmica. Brest *et al.* (2009) investigaram um algoritmo DE auto adaptativo (jDE), onde os parâmetros de controle F e CR são auto adaptados e um método multipopulação com mecanismo de envelhecimento é usado para lidar com *fitness*

landscapes dinâmico. Este algoritmo alcançou a primeira classificação na competição em "Computação Evolucionária em Ambientes Dinâmicos e Incertos" na CEC2009 (LI *et al.*, 2008). Angira e Santosh (2007) propuseram um algoritmo de evolução diferencial trigonométrico, baseado no esquema de mutação trigonométrica de Fan e Lampinen (2003) para resolver problemas de otimização dinâmica encontrados em Engenharia Química.

Nosso trabalho se assemelha ao de Shen *et al.* (2016), devido ao fato de ter como base o modelo proposto por eles. Porém, as principais diferenças em relação ao trabalho de Shen *et al.* (2016) é que uma extensão do modelo foi proposta para tentar se aproximar mais do mundo real, com a possibilidade de mais eventos dinâmicos, a adição de mais um objetivo e a possibilidade da ocorrência de eventos paralelos. Além disso, um algoritmo que ainda não foi utilizado para o DSPSP (CMODE) foi utilizado para a execução dos experimentos e uma comparação com o NSGA-III, considerado o algoritmo estado da arte, foi realizada.

3 Proposta para o DSPSP

Neste capítulo são apresentados o modelo proposto, que é uma extensão do modelo de Shen *et al.* (2016). Em seguida, os algoritmos de otimização aplicados ao modelo são discutidos. Por fim, o *framework* desenvolvido para a execução dos experimentos, bem como seus principais componentes são descritos.

3.1 O modelo proposto para o DSPSP

A modelagem para o DSPSP aqui proposta é uma extensão do modelo de Shen *et al.* (2016). A extensão inclui novos eventos dinâmicos, considera a ocorrência de eventos paralelos, além de estabelecer mais uma função objetivo, que incorpora a experiência da equipe ao modelo. O tomador de decisão também foi alterado para atender a adição do novo objetivo. A seguir, os principais elementos do modelo estendido são descritos, em conformidade com a ordem de descrição da Seção 2.3. Elementos como incorporação de incertezas (Seção 2.3.1), propriedades dos empregados (Seção 2.3.3) e restrições (Seção 2.3.7) não foram alterados em relação ao trabalho de Shen *et al.* (2016).

3.1.1 Incorporação de eventos dinâmicos

Os eventos dinâmicos adicionados ao modelo estendido foram:

(4) **Chegada de novos empregados**: No decorrer do projeto é comum a contratação de novos empregados. Uma vez que um novo empregado seja contratado, ele fica disponível para ser alocado nas tarefas.

(5) **Remoção de tarefas**: Com as mudanças de requisitos, pode acontecer de uma determinada tarefa ou conjunto de tarefas não seja mais necessária para o projeto ou as tarefas podem ser retiradas para reduzir o escopo. Quando uma tarefa é removida, todas as tarefas que dependem dela também são removidas em cascata.

Além disso, outro ponto importante que precisa ser destacado, é que no modelo estendido os eventos podem ocorrer simultaneamente, ou seja, eles podem acontecer em blocos, com vários eventos sendo ativados ao mesmo tempo.

3.1.2 Propriedades das tarefas

Em relação as propriedades das tarefas, uma nova variável foi adicionada para atender ao evento de remoção de tarefas, $\tau_j^{removida}$, que é uma variável binária que indica se a tarefa τ_j foi removida.

O processo de atualização do TPG foi alterado, uma vez que quando uma tarefa é removida ela não pode mais aparecer no TPG. Além disso, todas as tarefas que dependiam da tarefa removida também são removidas. A Figura 3.1 ilustra um exemplo de atualização do TPG quando uma tarefa é removida. Nesse caso, a tarefa τ_2 é removida e consequentemente a tarefa τ_6 é removida também, uma vez que dependia da tarefa τ_2 para ser feita. Mesmo que uma tarefa dependa parcialmente de uma tarefa removida, ela também será removida, por exemplo, caso a tarefa τ_3 fosse removida, a tarefa τ_5 também seria.

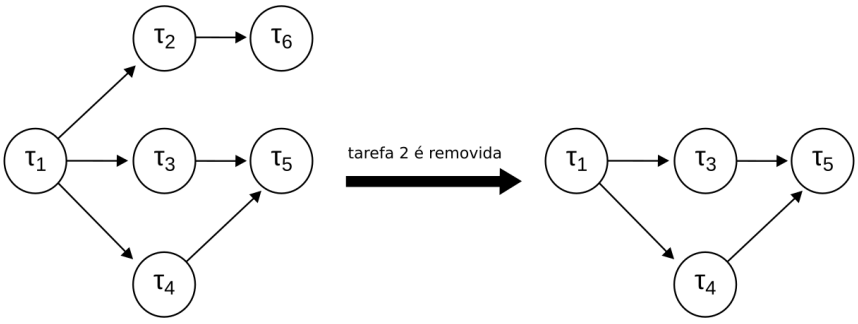

Figura 3.1 – Exemplo de atualização do grafo de precedência de tarefas (TPG), ao remover uma tarefa

3.1.3 Representação da solução

Devido ao novo evento dinâmico de remoção das tarefas, os valores da matriz de dedicação que devem ser levados em consideração pelos métodos de otimização foram alterados para: $md_{ij}(t_l) \in \{md_{ij}(t_l) | e_i^{disponivel}(t_l) = 1 \wedge \tau_j^{disponivel}(t_l) = 1 \wedge \tau_j^{removida} = 0\}$.

3.1.4 Objetivos a serem otimizados

Além dos objetivos descritos na Seçao 2.3.6, um novo objetivo foi adicionado ao modelo, como mostrado na Tabela 3.1

Tabela 3.1 – Novo objetivo a ser otimizado (maximizado

Função	Objetivo	Descrição
$f_5(t_l)$	Habilidade	Medida do nível das habilidades dos empregados que desenvolveram o software entre o instante t_l e o instante anterior.

A habilidade se refere ao nível de proficiência dos empregados para a realização das tarefas. Isso quer dizer que quanto maior o nível de experiência da habilidade melhor será a qualidade do software produzido. O objetivo aqui é atribuir os empregados às tarefas. Essa atribuição é baseada nos níveis de proficiência dos empregados para as habilidades requeridas pela tarefa específica. Para isso, o cálculo leva em consideração a média da proficiência dos empregados disponíveis que possuem a habilidade requerida por uma tarefa τ_j, disponível no instante t_l. Além disso, quanto maior a dedicação do empregado à tarefa, em conjunto com um maior nível de proficiência, melhor será a qualidade do software. Sendo assim, o valor total das habilidades requeridas para o projeto é computado pela soma desses valores, então:

$$\forall e_i \in E\_disp(t_l) \wedge \tau_j \in \mathrm{T}\_disp(t_l)$$

$$E_j^{prof} = \{e_i \in E\_disp(t_l) | e_{ij}^{proficiencia} > 0\}$$

$$\overline{E_j^{prof}} = \frac{\sum_{e_i \in E_j^{prof}} e_{ij}^{proficiencia}}{|E_j^{prof}|}$$

$$f_5(t_l) = habilidade = \sum_{\tau_j \in \mathrm{T\_}disp(t_l)} \sum_{e_i \in E\_disp(t_l)} \left(e_{ij}^{proficiencia} - \overline{E_j^{prof}} \right) \cdot md_{ij}(t_l)$$

3.1.5 Tratamento das restrições

No modelo estendido, uma penalidade foi adicionada ao tratamento da restrição das habilidades da tarefa, referente ao objetivo adicionado. O valor da penalização de f_5 é sempre menor do que o mínimo valor correspondente de qualquer solução viável do cronograma em t_l, dada por:

$$f_5(t_l) = habilidade = \left(\sum_{\tau_j \in \mathrm{T\_}disp(t_l)} \tau_j^{max\_emp} \cdot (-5) \right) \cdot reqsk$$

onde o valor 5, é referente ao valor máximo de proficiência que um empregado pode ter.

3.1.6 Tomador de decisões

O tomador de decisões (seção 2.3.9) foi modificado para acrescentar o objetivo adicionado. Sendo assim, $N\_o = 5$, ficando com um total de $N\_o \cdot (N\_o - 1)/2 = 5(5-1)/2 = 10$ comparações para a tomada de decisão.

3.2 Variantes do CMODE aplicadas ao DSPSP

O CMODE foi o algoritmo escolhido para ser modificado e testado no DSPSP. A principal razão do uso deste algoritmo é que ele tem como base o uso de duas técnicas aplicadas com sucesso na otimização com muitos objetivos: decomposição e múltiplas populações, onde cada subpopulação otimiza apenas um objetivo. Além disso, o uso de múltiplas populações tem apresentado bons resultados em problemas com muitos objetivos, além de ser uma maneira de garantir diversidade, uma característica útil no contexto de otimização dinâmica. Outro fator positivo é que o CMODE é um algoritmo que possui parâmetros auto adaptativos, uma técnica usada para lidar com problemas de otimização dinâmica.

Três variações do algoritmo CMODE foram testadas, são elas: o próprio CMODE; o CMODESDE, que é a implementação do CMODE+SDE; e o CMODESDENorm, que é uma variação do CMODESDE onde são normalizadas as distâncias entre os vizinhos mais próximos.

3.2.1 O CMODE aplicado ao DSPSP

O DSPSP caracteriza-se por ser um problema de otimização dinâmica com muitos objetivos e podemos utilizar o CMODE para resolvê-lo. Para isso, foi desenvolvido uma adaptação do modelo que simula o ambiente de execução do projeto de software. As informações de uma instância de projeto de software são passadas como parâmetro de entrada da simulação. Conforme mostrado na Seção 2.3, essas informações contemplam: a quantidade de empregados e suas habilidades; a quantidade de tarefas e as habilidades requeridas para elas; e os eventos dinâmicos que podem ocorrer ao longo do projeto, podendo mais de um evento ocorrer ao mesmo tempo, incluindo tipo e instante da ocorrência do evento.

A simulação de execução se inicia com o conjunto de empregados que precisam ser alocados ao conjunto de tarefas. Antes do início do projeto, é preciso montar um cronograma que minimize a duração e o custo, ao mesmo tempo que garanta que ele seja robusto quanto às incertezas provenientes das estimativas de esforço, além de levar em consideração as habilidades dos empregados, uma vez que elas podem afetar o cronograma. Sendo assim, o escalonamento inicial é realizado, consistindo em uma primeira execução do algorimto no instante t_0. Na primeira execução, apenas quatro objetivos são avaliados: duração, custo, robustez e habilidade. A estabilidade não é avaliada na primeira execução, pois seu cálculo depende da existência de cronogramas anteriores.

Nessa primeria execução do algoritmo, um conjunto de soluções não-dominadas que representam possíveis cronogramas que foram otimizados para serem adotados no início do projeto de software. A decisão sobre qual das soluções será selecionada é tomada conforme procedimento descrito nas Seções 2.3.9 e 3.1.6, gerando assim o cronograma inicial.

Com o cronograma inicial, o desenvolvimento do projeto de software é iniciado,

se encerrando quando todas as tarefas tiverem sido concluídas. No entanto, a execução do projeto está sujeita a eventos dinâmicos que foram passados como parâmetro das configurações da instância. Esses eventos causam o reescalonamento do projeto, e vale ressaltar que a simulação do projeto é composta por várias execuções diferentes do algoritmo.

Como mencionado anteriormente, a ocorrência de um ou mais eventos em um dado instante t_l implica na necessidade de um reescalonamento. Esses reescalonamentos consistem em novas execuções do algoritmo, sendo aplicados aos empregados e tarefas disponíveis em t_l. Uma vez executado o algoritmo, as soluções geradas passam pelo processo de tomada de decisão, onde um novo cronograma é escolhido para ser seguido. O desenvolvimento do projeto prossegue, sujeito aos eventos dinâmicos e seus respectivos reescalonamentos, até que todas as tarefas tenham sido finalizadas.

3.2.1.1 Estratégias dinâmicas incorporadas ao CMODE

O algoritmo CMODE foi modificado para explorar as características dinâmicas do DSPSP para obter melhor desempenho na otimização. A cada ponto de reescalonamento essas estratégias dinâmicas são utilizadas para tentar melhorar as soluções encontradas. As abordagens dinâmicas aplicadas ao CMODE foram: a detecção de mudanças, através da ocorrência dos eventos, as soluções são reparadas; e abordagens de memória, através do uso de soluções históricas.

Essas abordagens consistem em construir o arquivo populacional contendo inicialmente as soluções históricas. Pressupõe-se que soluções bem avaliadas em um instante anterior podem estar próximas das soluções desejadas para o instante atual. Além disso, é possível reparar soluções históricas de acordo com as características dos eventos dinâmicos ocorridos. Em um ponto de reescalonamento, essas soluções reparadas podem ser usadas para compor a população inicial do arquivo populacional da nova execução do algoritmo. Esse ajuste proativo consiste na informação do que mudou no espaço de busca em decorrência do evento dinâmico, sendo possível antecipar-se a essa mudança promovendo ajustes na solução corrente. Esse ajuste utiliza as informações históricas como base para a criação de uma solução reparada, que atenda ao novo estado do espaço de busca. Esses ajustes são aplicados quando ocorrem eventos de saída ou retorno de empregados, bem como na chegada de novos empregados.

Uma outra abordagem, utilizando a estratégia de memória, foi a criação de um arquivo externo ao algoritmo que guarda todas as soluções escolhidas a cada execução pelo tomador de decisão. Estas soluções arquivadas são utilizadas no passo da execução do DE para o arquivo populacional, inserindo um indivíduo desse arquivo externo para a geração do vetor de mutação. Para isso, uma alteração foi feita na linha 12 do Algoritmo 2, onde E_r representa um indivíduo aleatório do arquivo externo:

$$V_i = X_{i_1} + F_i \cdot (X_{i_2} - X_{i_3}) + F_i \cdot (E_r - X_{i_1})$$

Outra abordagem utilizada foi reavaliar as soluções desse arquivo externo a cada execução do algoritmo. Além disso, a junção de algumas dessas estratégias também é possível.

3.3 Estratégias dinâmicas aplicadas ao NSGA-III

Durante os experimentos foi preciso implementar estratégias dinâmicas no NSGA-III para comparar com o CMODE dinâmico. Como o foco deste trabalho não é o NSGA-III, foram utilizadas as mesmas estratégias usadas por Amaral (2018), que já estavam implementadas no *framework* e precisariam de poucos ajustes para adaptá-las para o NSGA-III. Essas estratégias utilizadas no NSGA-III consistem em construir a população inicial da busca utilizando o conhecimento armazenado. O algoritmo tem a possibilidade de incorporar três estratégias heurísticas para a inicialização de suas populações, utilizando abordagem de memória, abordagem preditiva e introdução de diversidade, são elas:

- **Uso de informações históricas**: cada evento dinâmico introduz mudanças no espaço de busca, podendo-se aproveitar as alocações de dedicação dos empregados obtidas pelo processo do reescalonamento anterior. No nosso trabalho isso é feito por meio da incorporação de algumas das soluções não-dominadas encontradas pela execução no instante t_{l-1}.
- **Ajuste proativo**: cada evento dinâmico consiste na informação do que mudou no espaço de busca, sendo possível antecipar-se a essa mudança promovendo ajustes na solução corrente. O ajuste proativo consiste em utilizar o cronograma vigente

como base para a criação de uma solução reparada que atenda ao novo estado do espaço de busca. O ajuste é aplicado quando ocorrem eventos de saída ou retorno de empregados, bem como na chegada de novos empregados.

- **Aleatoriedade**: para garantir diversidade às soluções, uma quantidade de soluções aleatórias são geradas. Essa quantidade varia de acordo com a parametrização de soluções históricas e do ajuste proativo.

3.4 *Framework* spsp-jmetal

Para avaliar o algoritmo proposto, foi desenvolvida uma extensão do *framework* apresentado por Amaral (2018), chamado spsp-jmetal[1]. Nessa extensão, algumas classes foram refatoradas para atender melhor aos modelos, sendo possível escolher executar os experimentos no modelo da Shen *et al.* (2016) que já existia, ou no modelo estendido, proposto por este trabalho. A finalidade do spsp-jmetal é servir como um *framework* de experimentos de meta-heurísticas populacionais aplicadas ao SPSP e DSPSP. A versão 5.2 do *framework* jMetal (DURILLO; NEBRO, 2011) foi utilizada para a execução dos algoritmos nos experimentos dos modelos. A escolha do jMetal deve-se a sua ampla utilização em estudos relacionados a meta-heurísticas aplicadas a problemas de otimização, dado que incorpora as implementações dos principais algoritmos utilizados na área de otimização.

A extensão do spsp-jmetal implementa o modelo do DSPSP descrito na seção 3.2. A implementação em Java utilizada neste trabalho construiu o modelo de acordo com os padrões de projeto predefinidos no jMetal 5.2. A interface *Problem* da biblioteca do jMetal é implementada pela classe *DSPSProblem* do spsp-jmetal, que por sua vez é estendida pela classe *DSPSProblemExtended*, permitindo que os algoritmos disponíveis no jMetal possam ser aplicados ao DSPSP sem necessidade de ajustes adicionais.

Para facilitar operações relacionadas à matriz de dedicação, as soluções são encapsuladas pela classe *DedicationMatrix*, que codifica o vetor unidimensional de números reais *DoubleSolution* do jMetal, em um objeto que encapsula uma matriz

[1] https://github.com/rodrigoamaral/spsp-jmetal

bidimensional de números reais e as operações necessárias para acessar os valores de dedicação de cada empregado para cada tarefa.

Um dos objetivos do spsp-jmetal é facilitar a implementação de experimentos nos quais diferentes algoritmos aplicados ao SPSP e suas variações são comparados entre si. Para isso, uma classe *ExperimentRunner* recebe por parâmetro o caminho para um arquivo no formato JSON com informações sobre a parametrização do experimento, incluindo uma lista de algoritmos a serem executados, total de vezes que esses algoritmos serão executados para cada instância, o número de avaliações da função objetivo, além de uma lista com o caminho para os arquivos de configuração de cada instância a ser avaliada. Os arquivos de configuração das instâncias também estão no formato JSON, tendo sido convertidos a partir do gerador de instâncias modificado que foi utilizado em (SHEN *et al.*, 2016), para incorporar as extensões do modelo aqui propostas.

Depois que a configuração do experimento é carregada em uma instância de classe *ExperimentSettings*, as execuções dos algoritmos são iniciadas. Cada execução é controlada pela classe *ExperimentRunner*, que utiliza a classe *AlgorithmAssembler* para construir as instâncias de execução de cada algoritmo a partir dos seus respectivos parâmetros de configuração.

As informações sobre tudo que envolve o projeto sendo simulado são controladas e armazenadas em instâncias das classes *Project*, *DynamicProject* e *DynamicExtended-Project* para SPSP e DSPSP, sendo as duas últimas somente para o DSPSP. Entre as operações que essas classes contemplam estão as avaliações das funções objetivos de cada problema. As principais classes mencionadas podem ser visualizadas no diagrama da Figura 3.2.

Cada grupo de eventos dinâmicos (eventos paralelos) que ocorrem na simulação de uma instância, dispara uma nova execução do algoritmo em experimentação no momento. Ao final de cada execução são gerados três arquivos: um com os vetores das soluções não-dominadas, outro com os valores das avaliações das funções objetivo para cada solução e outro com os valores normalizados das funções objetivo.

O spsp-jmetal conta com um mecanismo de *log* com *timestamp* e configurável quanto ao nível de severidade (*info, debug* ou *trace*). Esse recurso é importante principalmente no caso do DSPSP, devido à complexidade do modelo, é preciso ter um meio de

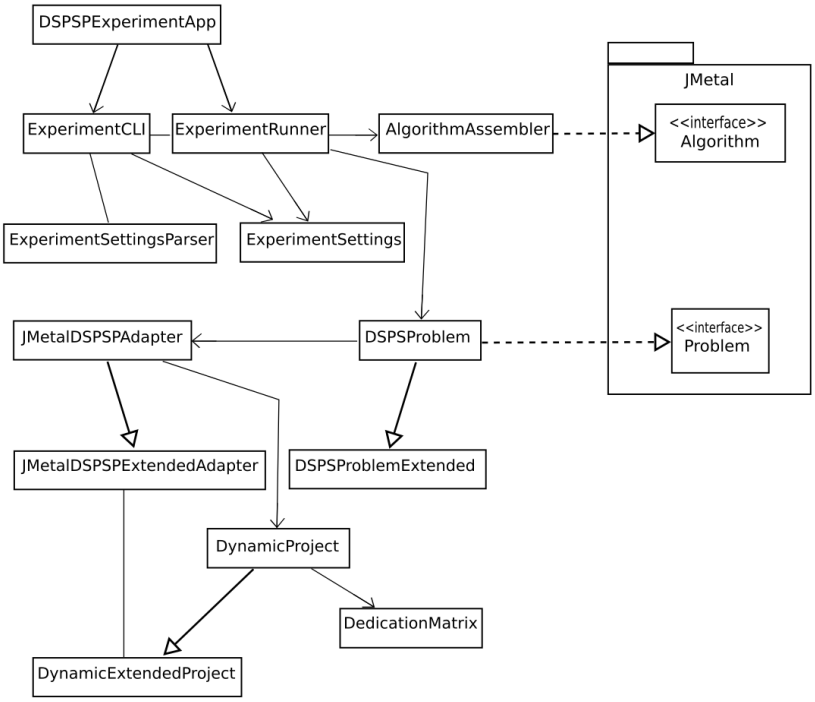

Figura 3.2 – Diagrama das principais classes do *framework*

verificar o andamento da execução e, principalmente, as informações sobre a simulação do projeto em cada ponto do reescalonamento.

4 Experimentos

Neste capítulo são apresentados os resultados dos experimentos feitos para avaliar o modelo proposto e o algoritmo utilizado. Na seção 4.1 é apresentado o planejamento do experimento, onde são descritas em quais condições foram feitos os experimentos, os parâmetros e instâncias utilizadas, além das questões de pesquisa e métricas. Por fim, a seção 4.2 discute os resultados dos experimentos.

4.1 Planejamento do Experimento

Para conduzir os experimentos, elaborou-se um planejamento onde foram realizadas as seguintes atividades:

- Definição do aparato experimental;
- Definição dos parâmetros dos algoritmos;
- Geração das instâncias de teste;
- Definição das questões de pesquisa;
- Definição das métricas de desempenho;
- Definição dos testes estatísticos de comparação dos resultados.

As seções a seguir discorrem sobre cada uma destas atividades.

4.1.1 Aparato Experimental

Para a execução de todos os experimentos foram utilizadas seis máquinas, que permitiram a execução paralela de diversos experimentos. Quatro das seis máquinas possuem a seguinte configuração: são máquinas virtuais com dois processadores Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz com 10GB de memória e rodam o sistema operacional Linux Debian 9 64Bits. Uma máquina possui processador Intel(R) Core(TM)

i3-3240 CPU @ 3.40GHz com 16GB de memória e roda o sistema operacional Linux Ubuntu 18.04 64Bits. A última máquina tem um processador Intel(R) Core(TM) i7-4510U CPU @ 2.00Ghz 2.60GHz com 8GB de memória e roda o sistema operacional Windows 10 64Bits. A versão do JAVA utilizada em todas as máquinas foi a Oracle JDK 1.8.

4.1.2 Parametrização

Para cada instância do problema foram realizadas 30 execuções de cada algoritmo que foi avaliado. Para garantir uma comparação justa entre os algoritmos, a execução de cada um deles realiza 20.000 avaliações da função objetivo. Então, calcula-se a média das métricas de qualidade obtidas para cada evento e em seguida é feita uma análise baseada em testes estatísticos.

Para cada algoritmo, calcula-se as médias dos primeiros 60 eventos de cada amostra. Essa decisão foi tomada devido à grande variação na quatidade de eventos que podem ocorrer até o final da simulação do projeto de cada instância. Como os eventos de cada instância ocorrem sempre na mesma ordem, entende-se que não há prejuízo em termos de comparação de algoritmos dentro da mesma instância.

Cada subpopulação do CMODE foi configurada com 20 indivíduos e o tamanho máximo do arquivo populacional foi configurado com 100 indivíduos. Para o NSGA-III, além dos paramêtros acima, também foram configurados os parâmetros de população inicial de 100 indivíduos, foi utilizado o operador *Simulated Binary Crossover*, para cruzamentos, com os valores de probabilidade de 0,9 e distribuição de 20, o operador de mutação polinomial com valores de probabilidade de $1/nv$, onde nv é o número de variáveis do problema, e distribuição de 20, e o operador de seleção de torneio binário.

4.1.3 Instâncias

Os experimentos consideraram 18 instâncias artificiais de simulação de projetos adaptadas ao modelo proposto, tomando por base o gerador de instâncias criado por Shen *et al.* (2016). Esse gerador foi adaptado para atender as novas informações do modelo utilizado neste trabalho. As instâncias se diferem pela quantidade de tarefas e de empregados iniciais, além das habilidades dos empregados. Em todas as instâncias há

um total de 10 habilidades requeridas para o projeto e cada tarefa requer 5 habilidades diferentes do conjunto das 10 habilidades. A quantidade de empregados iniciais de um projeto pode variar entre 5, 10 e 15. Além disso, o número de habilidades que cada empregado possui pode variar entre 4 e 5 em algumas instâncias e entre 6 e 7 em outras. Entre os empregados, 20% trabalham em tempo parcial, com uma dedicação que varia aleatoriamente no intervalo entre [0.5, 1); outros 20% podem trabalhar com possibilidade de horas extras, de modo que suas dedicações variam aleatoriamente no intervalo de (1, 1.5] e o restante trabalha tempo integral (dedicação igual a 1). Essas instâncias se diferem das usadas por Shen *et al.* (2016) nos seguintes aspectos: (*i*) dois novos tipos de eventos dinâmicos (remoção de uma tarefa e chegada de novo empregado) e (*ii*) possibilidade de ocorrência de eventos paralelos.

Cada instância é representada por um identificador que descreve suas características, seguindo o formato `sT#1_dT#2_sE#3_dE#4_SK#5-#6`. Nesse formato, #1 representa o número de tarefas iniciais do projeto, #2 representa o número de novas tarefas que são adicionadas durante o projeto, #3 representa o número inicial de empregados disponíveis no início do projeto, #4 representa o número de novos empregados que surgem durante o projeto, e #5 e #6 representam os números mínimo e máximo de habilidades que os empregados podem ter, respectivamente. Por exemplo, o identificador `sT20_dT10_sE10_dE1_SK4-5` denota que a instância possui 20 tarefas iniciais, que 10 novas tarefas chegam no decorrer do projeto, onde 10 empregados começam no projeto e 1 novo empregado chega durante o projeto, cada um possuindo entre 4 e 5 habilidades. Todas as instâncias utilizadas constam na Tabela 4.1. Em alguns casos o identificador será utilizado no lugar do nome da instância para facilitar a visualização das tabelas.

4.1.4 Métricas de Desempenho

Os resultados dos experimentos foram avaliados pelas métricas de hipervolume (FONSECA; PAQUETE; LÓPEZ-IBÁNEZ, 2006), comumente usado em trabalhos de otimização multiobjetivo; acurácia e estabilidade (CÁMARA; ORTEGA; TORO, 2007), que são características que podem ser consideradas para a avaliação da performance dos algoritmos em um processo de otimização dinâmica.

O hipervolume é calculado em relação a um ponto de referência no hiperplano, que deve representar uma solução ruim para não dominar nenhuma outra que faça parte da

Tabela 4.1 – Identificadores das instâncias utilizadas nos experimentos

Instâncias	Identificador
sT10_dT10_sE5_dE1_SK4-5	I1
sT10_dT10_sE10_dE1_SK4-5	I2
sT10_dT10_sE15_dE1_SK4-5	I3
sT10_dT10_sE5_dE1_SK6-7	I4
sT10_dT10_sE10_dE1_SK6-7	I5
sT10_dT10_sE15_dE1_SK6-7	I6
sT20_dT10_sE5_dE1_SK4-5	I7
sT20_dT10_sE10_dE1_SK4-5	I8
sT20_dT10_sE15_dE1_SK4-5	I9
sT20_dT10_sE5_dE1_SK6-7	I10
sT20_dT10_sE10_dE1_SK6-7	I11
sT20_dT10_sE15_dE1_SK6-7	I12
sT30_dT10_sE5_dE1_SK4-5	I13
sT30_dT10_sE10_dE1_SK4-5	I14
sT30_dT10_sE15_dE1_SK4-5	I15
sT30_dT10_sE5_dE1_SK6-7	I16
sT30_dT10_sE10_dE1_SK6-7	I17
sT30_dT10_sE15_dE1_SK6-7	I18

fronteira avaliada. Assim sendo, um maior valor do hipervolume indica que o algoritmo obteve um melhor conjunto de soluções. No contexto desse trabalho, hipervolumes altos indicam um melhor conjunto de escalonamentos, considerando todos os objetivos calculados. A Figura 4.1 ilustra um exemplo de como uma fronteira bidimensional, com seus pontos da fronteira ($p^{(1)}$, $p^{(2)}$, $p^{(3)}$), tem seu hipervolume obtido em relação a um ponto de referência r.

A acurácia mede a qualidade da solução encontrada. Ela quantifica a qualidade da solução como uma relação entre o hipervolume (HV) da PF^* e o hipervolume máximo (HV_{max}) que foi encontrado até o momento. Sendo assim, a acurácia é definida como:

$$acc(t) = \frac{HV(PF^*(t))}{HV_{max}(PF^*(t))}$$

Uma acurácia com o valor 1 ou próximo a 1 indica que o algoritmo alcançou o melhor resultado ou próximo ao melhor resultado no evento. Ela é importante para medir o desempenho em ambientes dinâmicos, pois espera-se que as técnicas de otimi-

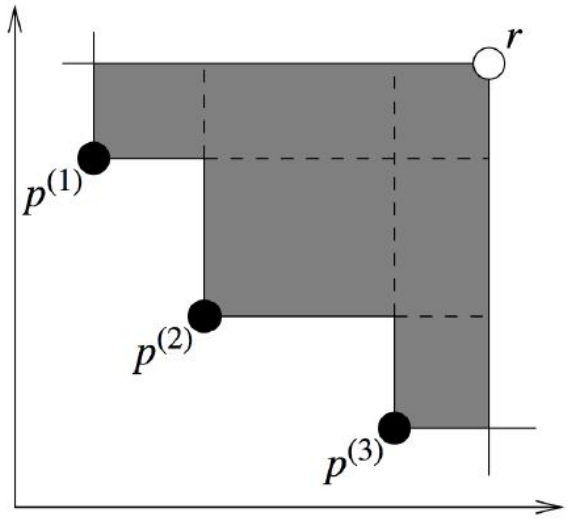

Figura 4.1 – Hipervolume de uma fronteira de Pareto para dois objetivos (adaptado de (FONSECA; PAQUETE; LÓPEZ-IBÁNEZ, 2006)

zação dinâmica façam o algoritmo utilizar o conhecimento já obtido para melhorar o desempenho em um novo evento.

No contexto de otimização dinâmica, um algoritmo adaptativo é chamado de estável, se as mudanças no ambiente não afetarem a precisão da otimização severamente. Mesmo no caso de mudanças drásticas, um algoritmo deve ser capaz de limitar a respectiva queda do desempenho. A estabilidade pode ser vista como um índice de consistência de quão confiável o algoritmo continua obtendo bons resultados durante todo o processo de otimização. Esta métrica nos dá uma ideia sobre o efeito das mudanças no ambiente na precisão do algoritmo. Em resumo, a estabilidade indica o quão bem um algoritmo se recupera após a ocorrência de uma mudança no ambiente. Sendo assim, a estabilidade é definida como:

$$stab(t) = max\{0, acc(t-1) - acc(t)\}$$

Uma estabilidade com o valor 0 significa que o algoritmo melhora, ou pelo menos não piora, a cada evento e se manteve estável as mudanças que ocorreram no ambiente. Sendo assim, um baixo valor de *stab* indica boa performance.

4.1.5 Testes Estatísticos

A comparação dos resultados leva em consideração a significância estatística apurada pelos testes de hipótese não-paramétricos de Wilcoxon, para comparação entre duas amostras, e de Friedman, para comparação simultânea entre três ou mais amostras (DERRAC *et al.*, 2011). Em todos os casos foram considerados um valor-p menor que 0,05 (o que indica um nível de confiança de mais de 95%).

O programa que foi utilizado e desenvolvido para o experimento recebe um arquivo de configuração no qual podem ser parametrizados: a quantidade de execuções de cada algoritmo, os caminhos para o arquivo de configuração das instâncias a serem executadas, os algoritmos a serem utilizados e seus parâmetros, assim como o modelo utilizado para a execução do mesmo. A análise dos resultados foi feita em *Python*, com o uso das bibliotecas: `SciPy`[1] para computação numérica e funções estatísticas, `pandas`[2] para análise de dados, `Matplotlib`[3] para construção de gráficos e `scikit-posthocs`[4] para os testes estatísticos *post hoc*.

4.1.6 Questões de Pesquisa

As questões de pesquisa que guiaram os experimentos foram:

- **QP1.** Como o modelo proposto se comporta em relação ao modelo original? Com o intuito de saber se o modelo proposto gera perdas em relação ao desempenho do algoritmo comparado com o modelo original, uma vez que o modelo proposto possui mais objetivos e novos eventos dinâmicos, além da possibilidade da ocorrência de eventos em paralelo, deixando-o mais próximo do mundo real. Para isso, foi feita uma análise qualitativa dos objetivos através da execução do CMODE para cada um dos modelos, considerando a instância I1, que é a mais simples do problema.

[1] https://www.scipy.org/scipylib/

[2] https://pandas.pydata.org/

[3] https://matplotlib.org/

[4] https://github.com/maximtrp/scikit-posthocs

- **QP2.** Qual variação do algoritmo base tem melhor desempenho aplicado ao DSPSP? Antes de avaliar o algoritmo CMODE e as suas modificações, foi necessário testar previamente algumas de suas variações bases. Para isso, foram definidas três configurações para o CMODE: uma com a implementação feita de acordo com o Algoritmo 1 que foi chamada de CMODE; o segundo algoritmo foi utilizando o SDE proposto por (WANG; ZHANG; ZHANG, 2016) para problemas com muitos objetivos (CMODESDE); e a última variação foi o CMODESDE com a distância normalizada (CMODESDENorm). Para essa questão, os algoritmos foram aplicados às instâncias `sT10_dT10_sE5_dE1_SK4-5`, `sT10_dT10_sE15_dE1_SK6-7`, `sT20_dT10_sE5_dE1_SK6-7`, `sT30_dT10_sE15_dE1_SK4-5`. O teste estatístico realizado foi o de Friedman e as métricas utilizadas foram as três apresentadas anteriormente.

- **QP3.** Como o desempenho do algoritmo CMODE se compara ao do NSGA-III quando aplicado ao DSPSP? Com o intuito de validar se o algoritmo CMODE pode ser efetivo quando aplicado ao DSPSP, foi realizado um comparativo entre o CMODESDE e um algoritmo considerado estado da arte sobre meta-heurísticas com muitos objetivos, o NSGA-III (DEB; JAIN, 2014). A ideia é comparar um algoritmo genético (mesma meta-heurística da abordagem da (SHEN *et al.*, 2016)) com uma abordagem de DE aplicada ao DSPSP. Esses algoritmos foram aplicados a todas as 18 instâncias, utilizando todas as métricas apresentadas e o teste de Wilcoxon para os testes estatísticos de comparação.

- **QP4.** Qual a influência de cada estratégia dinâmica no desempenho do algoritmo CMODE? Para compreender como cada uma das técnicas contribui para o desempenho do algoritmo, realizou-se um comparativo entre algumas variantes do CMODESDE descritos na Tabela 4.2. As variantes diferem de acordo com a forma de utilização do arquivo e da utilização do ajuste proativo. Além disso, uma variante do NSGA-III utilizando uma estratégia dinâmica para construção da população inicial foi utilizada. Essa estratégia utilizada no NSGA-III foi a mesma utilizada por (AMARAL, 2018), com as mesmas proporções da construção da população inicial utilizadas por (SHEN *et al.*, 2016). Essas variantes foram aplicadas a todas as 18 instâncias, sendo considerandos todas as três metricas apresentadas e com o teste estatístico de Friedman.

Tabela 4.2 – Variantes do CMODESDE

Variante	Arquivo Externo	Arquivo Externo reavaliado	Histórico	Ajuste Proativo
CMODESDEDynamic			x	
CMODESDEExternalDynamic	x			
CMODESDEExternalReDynamic		x		
CMODESDERepairDynamic				x
CMODESDEFullDynamic	x		x	
CMODESDEFullReDynamic		x	x	

4.2 Resultados

Nos gráficos utilizados para análise deste trabalho, o eixo *x* indica a sequência de pontos de reescalonamento do projeto, motivados pela ocorrência de eventos. Já o eixo *y* refere-se ao valor dos hipervolumes, acurácias e estabilidades. Cada ponto indica o valor de hipervolume/acurácia/estabilidade obtido pelo conjunto de soluções não-dominadas geradas pela execução de um determinado algoritmo em um determinado ponto do reescalonamento. Os gráficos referentes as instâncias da QP3 e QP4 que não foram discutidos no texto encontram-se no Apêndice A e B respectivamente.

4.2.1 Comparação entre os modelos

O objetivo da QP1 é verificar se a extensão do modelo apresenta alguma deterioração da solução, uma vez que foi adicionado um objetivo e mais eventos dinâmicos, além da ocorrência de eventos paralelos. Em suma, a ideia é verificar se os objetivos existentes no modelo de Shen *et al.* (2016) podem piorar na extensão do modelo proposto, que é mais complexo. O experimento foi executado com o algoritmo CMODE, considerando instâncias semelhantes (mesmo número de tarefas, habilidades e empregados iniciais) e a mais simples, a instância `sT10_dT10_E5_SK4-5` para o modelo usado por Shen *et al.* (2016) e a instância `sT10_dT10_sE5_dE1_SK4-5` para a extensão do modelo proposto.

Neste experimento, para efetuar a análise comparativa dos modelos, foram selecionadas três soluções extraídas da execução que apresentou o melhor hipervolume, ou seja, a melhor execução do algoritmo para cada modelo. Dentre as soluções extraídas

estão a primeira solução gerada, a última e uma solução do meio entre todas as soluções geradas da execução de melhor hipervolume.

Comparando os valores dos objetivos apresentados na Tabela 4.3, para o modelo original, com os da Tabela 4.4, para o modelo estendido, podemos perceber que a execução do CMODE obteve melhores resultados quando aplicado ao modelo proposto. A execução do algoritmo no modelo proposto apresentou melhores valores para os objetivos de Duração, Custo e Estabilidade. O modelo de Shen *et al.* (2016) só conseguiu obter melhores resultados no objetivo de robustez. Vale destacar que o modelo proposto ainda considera um objetivo extra, que é a experiência.

Tabela 4.3 – Valores dos objetivos obtidos pela execução do CMODE que obteve o melhor hipervolume considerando o modelo de Shen *et al.* (2016)

Solução	Duração	Custo	Robustez	Estabilidade
1ª	19,74323	349817,23804	0,39725	2,35525
50ª	14,25828	357761,05056	0,42076	1,89231
100ª	20,98193	535005,58159	0,25072	2,70073

Tabela 4.4 – Valores dos objetivos obtidos pela execução do CMODE que obteve o melhor hipervolume considerando o modelo proposto

Solução	Duração	Custo	Robustez	Experiência	Estabilidade
1ª	18,13265	207174,63666	1,72691	0,21067	0,298536
5ª	3,98058	64774,42334	1,72691	0,30937	0,78857
10ª	6,39066	126089,10746	1,72691	0,20089	0,85427

De maneira geral, o modelo proposto não apresentou deterioração em relação ao modelo original, pelo contrário, se mostrou até mais efetivo do que o modelo de Shen *et al.* (2016). A única ressalva em relação ao modelo original foi a prova em relação ao objetivo de robutez. Com isso, podemos concluir que tornamos o modelo mais complexo, e mais próximo do mundo real, sem que houvesse uma perda no desempenho do algoritmo. Isso mostra que a extensão do modelo proposto é viável para os algoritmos da literatura, quando comparado com o modelo de Shen *et al.* (2016). Contudo, um estudo não foi realizado para identificar por que a extensão do modelo proposto se saiu melhor que o modelo de Shen *et al.* (2016), uma vez que não está no escopo desse trabalho.

4.2.2 Comparação entre os algoritmos base

O experimento relativo a QP2 tem como objetivo comparar o desempenho entre três variações base do algoritmo CMODE, sendo elas: CMODE, CMODESDE, CMODESDENorm. Esses algoritmos foram executados em quatro instâncias: I1 (`sT10_dT10_sE5_dE1_SK4-5`), para observar o desempenho quando há poucos empregados e poucas tarefas; I6 (`sT10_dT10_sE15_dE1_SK6-7`), para avaliar o desempenho com poucas tarefas e muitos empregados; I10 (`sT20_dT10_sE5_dE1_SK6-7`), para verificar o desempenho com poucos empregados e uma quantidade mediana de tarefas; e I15 (`sT30_dT10_sE15_dE1_SK4-5`), para observar o comportamento dos algoritmos com muitas tarefas e muitos empregados.

Nas Tabelas 4.5, 4.6, 4.7 é exibido o valor médio e o desvio padrão (entre parênteses) dos hipervolumes, acurácias e estabilidades, respectivamente, dos algoritmos bases nas quatro instâncias. Os melhores resultados encontram-se realçados em cinza. Esta convenção será usada no restante deste documento em algumas tabelas.

Tabela 4.5 – Média e desvio padrão dos hipervolumes de cada algoritmo base em cada instância

Instância	CMODE	CMODESDE	CMODESDENorm
I1	1,20E+00 (3,09E-01)	1,18E+00 (3,05E-01)	1,20E+00 (3,09E-01)
I6	7,76E-01 (4,23E-01)	7,78E-01 (4,23E-01)	7,54E-01 (4,28E-01)
I10	9,14E-01 (3,77E-01)	9,02E-01 (3,69E-01)	9,09E-01 (3,75E-01)
I15	8,48E-01 (4,45E-01)	8,53E-01 (4,46E-01)	8,50E-01 (4,50E-01)

Tabela 4.6 – Média e desvio padrão das acurácias de cada algoritmo base em cada instância

Instância	CMODE	CMODESDE	CMODESDENorm
I1	8,67E-01 (1,64E-01)	8,64E-01 (1,63E-01)	8,69E-01 (1,64E-01)
I6	6,24E-01 (2,88E-01)	6,24E-01 (2,87E-01)	6,05E-01 (2,92E-01)
I10	8,31E-01 (2,00E-01)	8,29E-01 (1,99E-01)	8,30E-01 (2,00E-01)
I15	7,80E-01 (2,76E-01)	7,79E-01 (2,76E-01)	7,78E-01 (2,75E-01)

A Figura 4.2 apresenta os valores dos hipervolumes a cada ponto de reescalonamento para a instância I1. Nela, não fica claro quem possui valores mais altos. Porém, como podemos constatar na Tabela 4.5, os algoritmos CMODE e CMODESDENorm obtiveram a mesma média do hipervolume para essa instância, e o CMODESDE obteve

Tabela 4.7 – Média e desvio padrão das estabilidades de cada algoritmo base em cada instância

Instância	CMODE	CMODESDE	CMODESDENorm
I1	1,15E-01 (7,55E-01)	1,02E-01 (7,59E-01)	9,84E-02 (7,14E-01)
I6	5,40E-02 (4,19E-01)	7,85E-02 (6,06E-01)	7,42E-02 (5,61E-01)
I10	7,53E-02 (3,90E-01)	7,20E-02 (3,85E-01)	6,33E-02 (3,01E-01)
I15	6,97E-02 (5,44E-01)	8,56E-02 (6,70E-01)	6,54E-02 (5,56E-01)

um valor um pouco abaixo. Quanto mais alto o valor do hipervolume mais próximo do ótimo estarão as soluções encontradas. Na execução do pós-teste ao teste de Friedman, realizando uma comparação pareada dos valores-p, Tabela 4.8, confirmamos as impressões da visualização dos dados, indicando que não há diferença significativa no desempenho dos algoritmos para a instância I1.

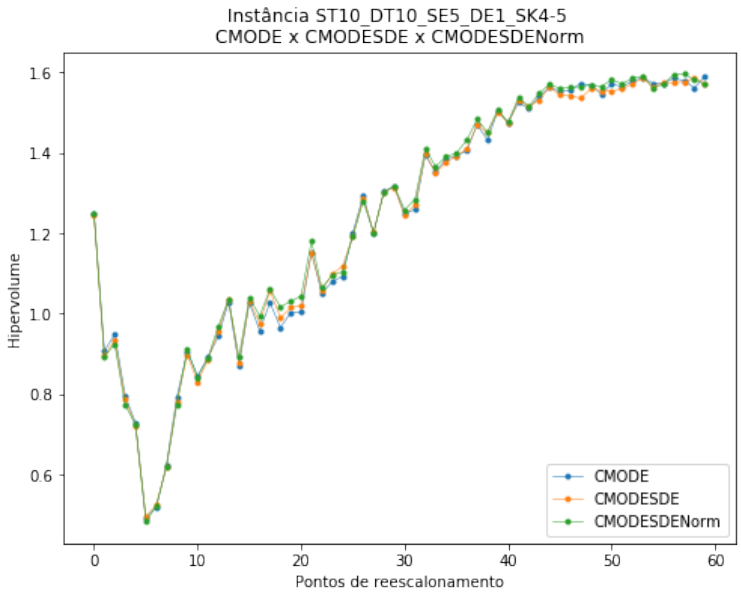

Figura 4.2 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos base para a instância I1

Observando a Figura 4.3, podemos notar que entre os pontos de reescalonamento 15 e 30, o CMODE obteve piores valores de acurácia em relação aos outros algoritmos, mas também não fica claro qual foi o melhor em relação a acurácia. De acordo com a Tabela 4.6, quem obteve a melhor média foi o CMODESDENorm. Quanto maior o valor

Tabela 4.8 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos base para a instância I1

	CMODE	CMODESDE	CMODESDENorm
CMODE	-1,00E+00	8,95E-01	8,13E-01
CMODESDE	8,95E-01	-1,00E+00	7,12E-01
CMODESDENorm	8,13E-01	7,12E-01	-1,00E+00

da acurácia, melhor o desempenho do algoritmo em ambientes dinâmicos. Isso indica que em média o CMODESDENorm consegue gerar melhores soluções a cada novo evento. O teste de Friedman realizado mostrou que para a acurácia não existe diferença estatística entre os algoritmos, obtendo um valor-*p* de 1,523833E-01.

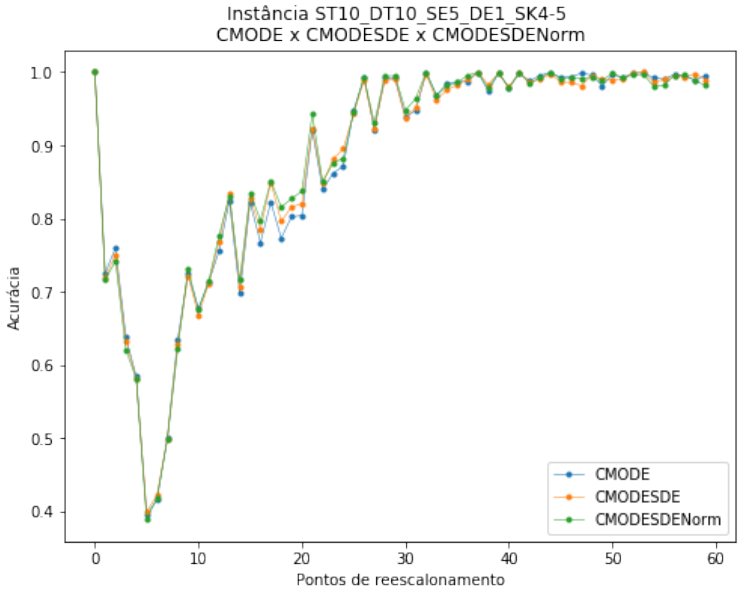

Figura 4.3 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos base para a instância I1

Levando-se em consideração a estabilidade, observando a Figura 4.4 podemos perceber que o algoritmo mais instável é o CMODE, obtendo os piores valores de estabilidade e que aparentemente o CMODESDENorm foi mais estável entre eles. Essa informação pode ser confirmada na Tabela 4.7. Quanto menor o valor da estabilidade, melhor a resposta do algoritmo em relação as mudanças no ambiente, tornando-o mais estável. O teste de Friedman constatou que não existe diferença estatística entre os

algoritmos considerando a estabilidade, possuindo um valor-*p* de 7,419827E-01.

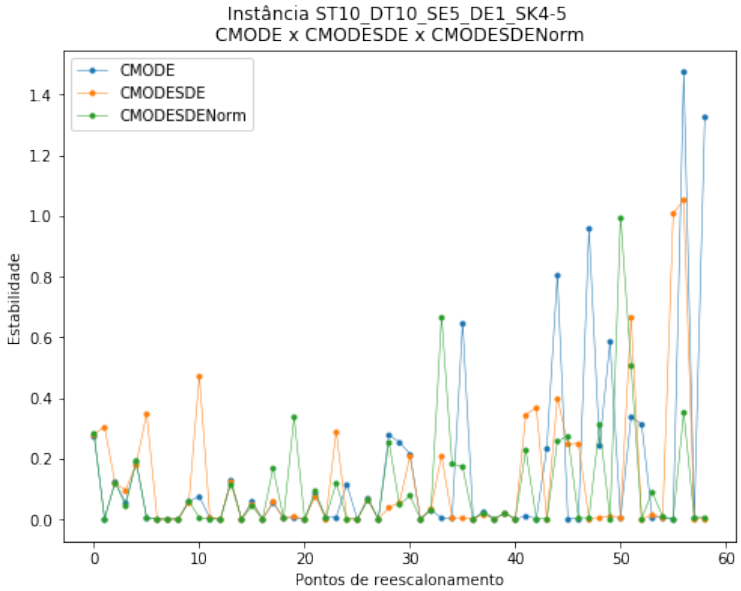

Figura 4.4 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos base para a instância I1

A Figura 4.5 apresenta os valores dos hipervolumes a cada ponto de reescalonamento para a instância I6. Neste caso percebe-se uma queda nos valores médios dos hipervolume para todas as variações do algoritmo. E como aconteceu na instância anterior, não fica fácil discernir qual algoritmo foi melhor. Entretanto, podemos visualizar que o CMODESDENorm está um pouco abaixo dos demais, logo ele foi pior que os outros. Na execução do pós-teste do teste estatístico, ao realizar uma comparação pareada, Tabela 4.9, é confirmado que não existe diferença significativa entre eles.

Tabela 4.9 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos base para a instância I6

	CMODE	CMODESDE	CMODESDENorm
CMODE	-1,00E+00	9,62E-01	4,55E-01
CMODESDE	9,62E-01	-1,00E+00	4,84E-01
CMODESDENorm	4,55E-01	4,84E-01	-1,00E+00

Para a métrica de acurácia, Figura 4.6, podemos perceber que entre os pontos de reescalonamento 40 e 50 o CMODE obtém valores um pouco melhor do que os outros

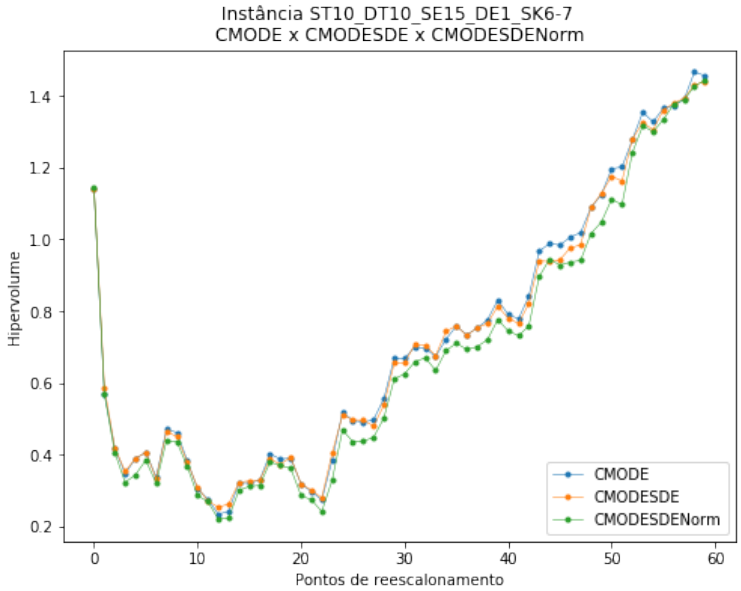

Figura 4.5 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos base para a instância I6

algoritmos, porém ao chegar no último ponto ele está ligeiramente abaixo dos demais. Na comparação pareada do pós-teste estatístico, nota-se que essa diferença não existe.

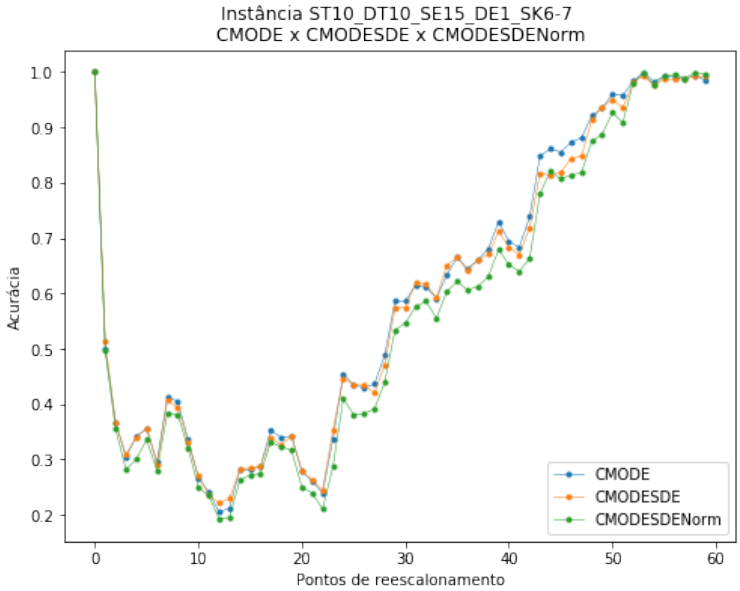

Figura 4.6 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos base para a instância I6

Com relação à estabilidade, ainda para a instância I6, podemos perceber pela Figura 4.7 uma instabilidade maior em relação a instância anterior, com o CMODE se mantendo mais estável que os demais algoritmos. Entretanto, o teste estatístico mostrou que esses dados possuem a mesma distribuição, obtendo um valor-*p* de 1,739654E-01.

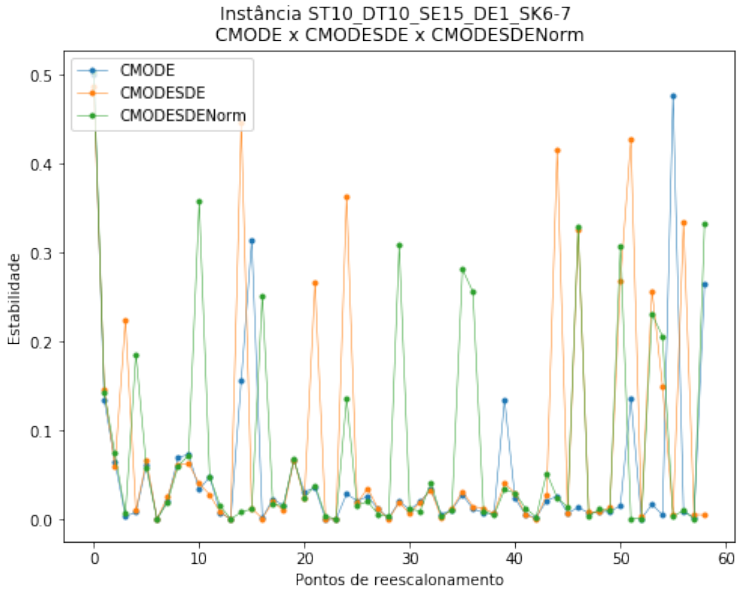

Figura 4.7 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos base para a instância I6

Na Figura 4.8, é apresentado os valores dos hipervolumes a cada ponto de reescalonamento para a instância I10. Nessa instância, nota-se uma equiparação entre os valores do hipervolume dos algoritmos com leves diferenças entre eles em cada ponto de reescalonamento. De acordo com a Tabela 4.5, o CMODE obteve um valor médio do hipervolume maior que os outros dois algortimos. Mais uma vez, como aconteceu nos casos anteriores, ao realizar uma comparação pareada entre os valores-*p* (Tabela 4.10), confirma aquilo que podemos perceber pelo gráfico, que não há diferença significativa entre os resultados obtidos pelos algoritmos.

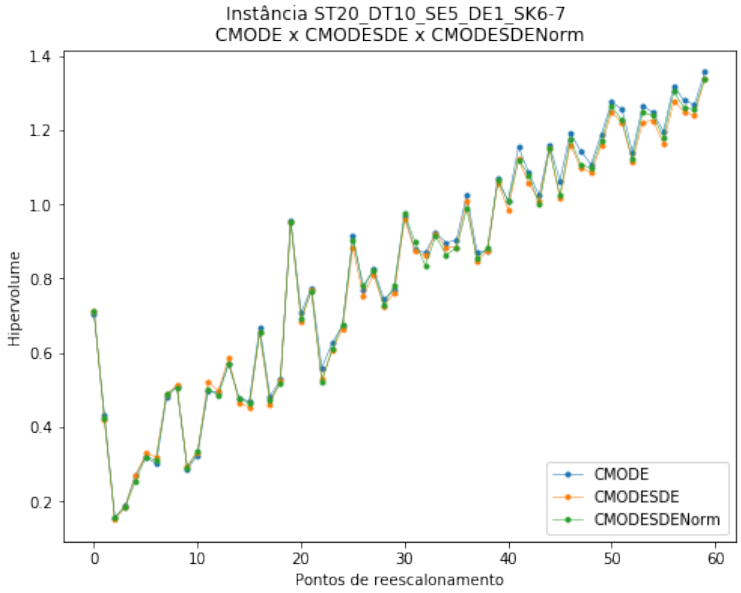

Figura 4.8 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos base para a instância I10

Tabela 4.10 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos base para a instância I10

	CMODE	CMODESDE	CMODESDENorm
CMODE	-1,00E+00	7,63E-01	8,37E-01
CMODESDE	7,63E-01	-1,00E+00	9,24E-01
CMODESDENorm	8,37E-01	9,24E-01	-1,00E+00

Considerando agora a acurácia para essa mesma instância, Figura 4.9, podemos ver que em alguns pontos de reescalonamento o CMODESDE obtém melhores resultados e em outros piores, entretanto não fica claro qual delas se saiu melhor. Porém, o CMODE foi o algoritmo que obteve melhor média, como constatado pela Tabela 4.6, com uma diferença mínima entre os outros algoritmos. O teste estatístico condiz com o que é visível no gráfico, mostrando que não possui diferença estatística entre eles, com um valor-*p* de 1,553509E-01.

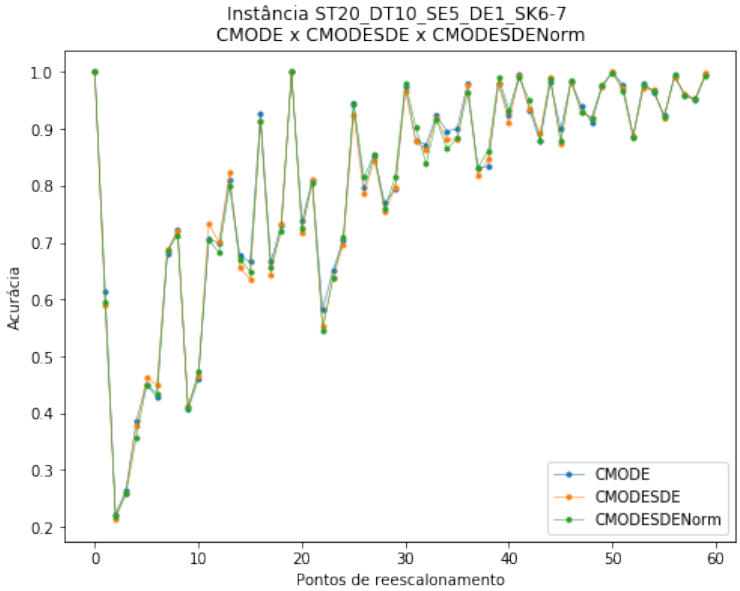

Figura 4.9 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos base para a instância I10

A estabilidade pode ser vista na Figura 4.10. O teste estatístico mostrou que não possui diferença estatística entre os algoritmos considerando a estabilidade, com um valor-*p* de 2,536349E-01.

O resultado das métricas de hipervolume, acurácia e estabilidade para a instância I15 podem ser visualizados nas Figuras 4.11, 4.12, 4.13, respectivamente. Na Figura 4.11, podemos perceber que o CMODESDE é o que se destaca em relação aos outros algortimos. Como aconteceu nas instâncias anteriores, ao realizar uma comparação pareada entre os valores-*p* (Tabela 4.11), fica constatado que não existe diferença estatística entre eles.

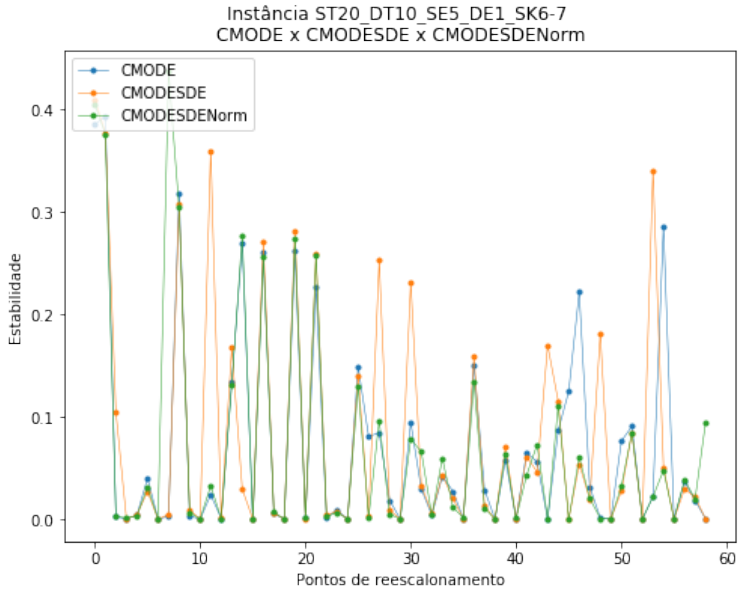

Figura 4.10 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos base para a instância I10

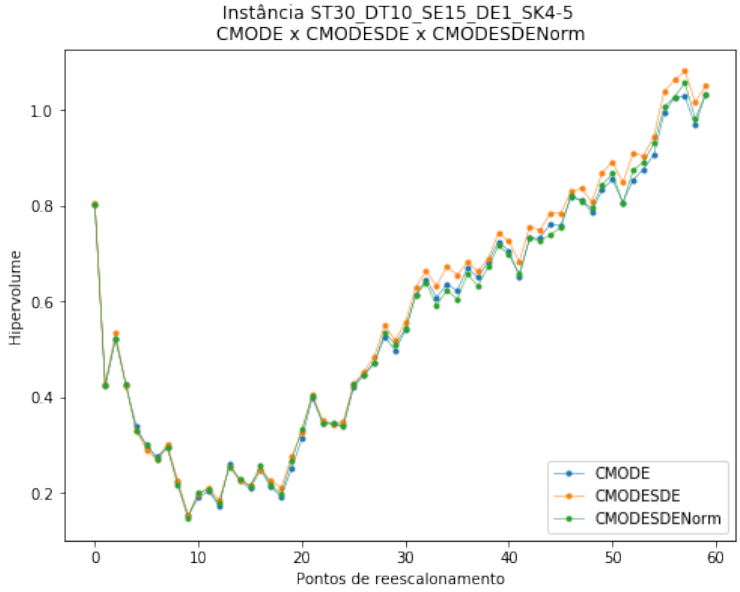

Figura 4.11 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos base para a instância I15

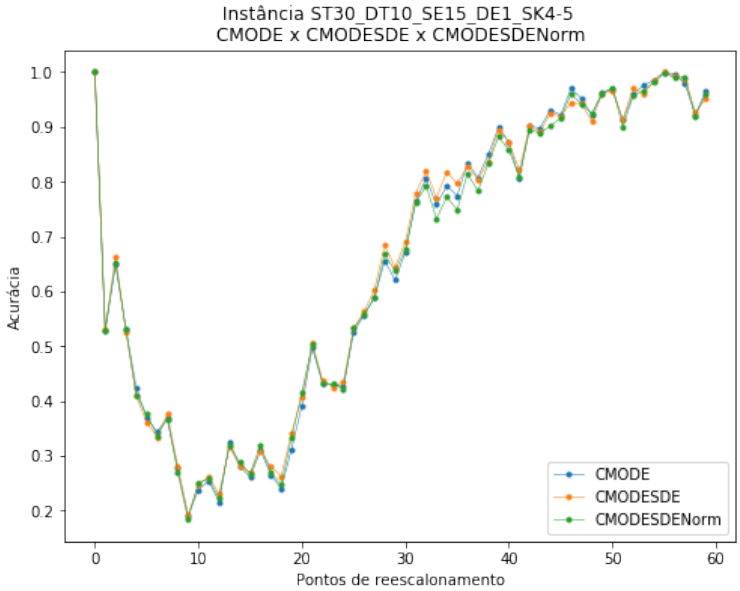

Figura 4.12 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos base para a instância I15

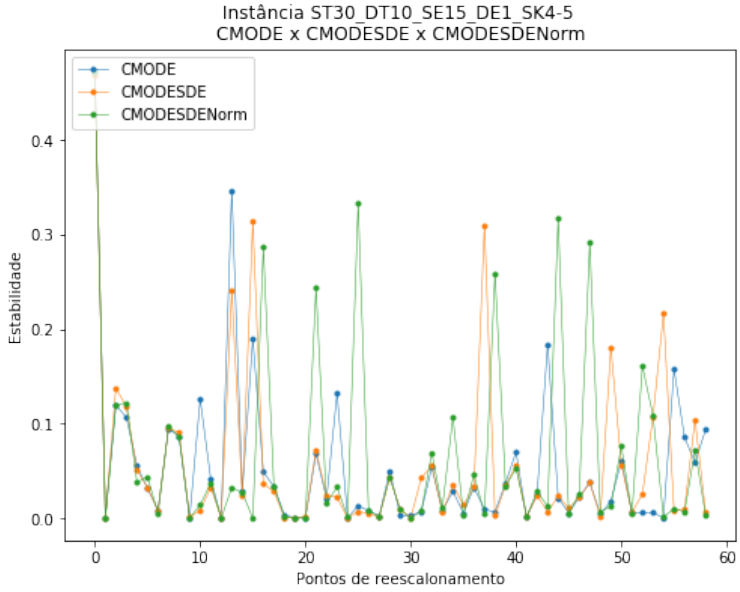

Figura 4.13 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos base para a instância I15

Os valores máximos que os algoritmos conseguem alcançar vão diminuindo à

Tabela 4.11 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos base para a instância I15

	CMODE	CMODESDE	CMODESDENorm
CMODE	-1,00E+00	6,84E-01	9,83E-01
CMODESDE	6,84E-01	-1,00E+00	6,99E-01
CMODESDENorm	9,83E-01	6,99E-01	-1,00E+00

medida que a quantidade de informações contidas nas instâncias aumenta. Além disso, o CMODE e CMODESDE obtiveram melhores resultados em duas das quatro instâncias considerando o hipervolume. Já levando em consideração a acurácia, o CMODE teve uma média melhor em duas instâncias e o CMODESDENorm foi o que obteve melhor resultado em relação à estabilidade em três das quatro instâncias. Vale destacar que em todas as instâncias, considerando todas as métricas, os pós-testes estatísticos constataram o que se vê nas imagens, que não existe diferença estatística entre os algoritmos. Apesar do CMODE ter se saído melhor considerando a acurácia (2 das 4 instâncias) e empatado com o CMODESDE considerando o hipervolume (2 das 4 instâncias), o algoritmo que será utilizado nos experimentos seguintes será o CMODESDE por ser um algoritmo voltado para problemas com muitos objetivos e só ter perdido para o CMODESDENorm na estabilidade.

4.2.3 Comparação com o NSGA-III

Após a comparação entre as variações do algoritmo base, escolheu-se o CMODESDE para ser comparado com o NSGA-III. Essa comparação, referente ao experimento realizado para a QP3, tem o objetivo de investigar se o CMODESDE é competitivo em relação ao algoritmo considerado estado da arte para muitos objetivos. Em resumo, os algoritmos originais são testados para saber quem tem melhor desempenho.

A Tabela 4.12, apresenta a média e desvio padrão do hipervolume do CMODESDE e NSGA-III para todas as instâncias. O CMODESDE obteve valores mais altos em 14 das 18 instâncias, sendo que na instância I5 os algoritmos obtiveram as mesmas médias.

Tabela 4.12 – Média e desvio padrão dos hipervolumes do CMODESDE e NSGA-III em cada uma das 18 instâncias

Instância	CMODESDE	NSGA-III
I1: sT10_dT10_sE5_dE1_SK4-5	1,15E+00 (3,19E-01)	1,18E+00 (3,09E-01)
I2: sT10_dT10_sE10_dE1_SK4-5	8,49E-01 (3,90E-01)	8,35E-01 (3,95E-01)
I3: sT10_dT10_sE15_dE1_SK4-5	9,18E-01 (3,35E-01)	8,79E-01 (3,53E-01)
I4: sT10_dT10_sE5_dE1_SK6-7	8,99E-01 (3,88E-01)	8,98E-01 (3,92E-01)
I5: sT10_dT10_sE10_dE1_SK6-7	9,12E-01 (4,06E-01)	9,12E-01 (4,19E-01)
I6: sT10_dT10_sE15_dE1_SK6-7	7,71E-01 (4,26E-01)	7,70E-01 (4,23E-01)
I7: sT20_dT10_sE5_dE1_SK4-5	8,35E-01 (3,96E-01)	8,32E-01 (4,01E-01)
I8: sT20_dT10_sE10_dE1_SK4-5	9,19E-01 (4,37E-01)	8,94E-01 (4,58E-01)
I9: sT20_dT10_sE15_dE1_SK4-5	8,46E-01 (4,31E-01)	8,08E-01 (4,47E-01)
I10: sT20_dT10_sE5_dE1_SK6-7	9,19E-01 (3,65E-01)	9,25E-01 (3,75E-01)
I11: sT20_dT10_sE10_dE1_SK6-7	8,09E-01 (4,37E-01)	7,85E-01 (4,50E-01)
I12: sT20_dT10_sE15_dE1_SK6-7	8,13E-01 (4,24E-01)	7,87E-01 (4,33E-01)
I13: sT30_dT10_sE5_dE1_SK4-5	7,16E-01 (3,22E-01)	7,00E-01 (3,25E-01)
I14: sT30_dT10_sE10_dE1_SK4-5	9,15E-01 (3,77E-01)	8,81E-01 (3,96E-01)
I15: sT30_dT10_sE15_dE1_SK4-5	8,47E-01 (4,44E-01)	7,98E-01 (4,56E-01)
I16: sT30_dT10_sE5_dE1_SK6-7	8,05E-01 (3,55E-01)	7,68E-01 (3,61E-01)
I17: sT30_dT10_sE10_dE1_SK6-7	8,12E-01 (4,42E-01)	7,88E-01 (4,57E-01)
I18: sT30_dT10_sE15_dE1_SK6-7	8,32E-01 (4,61E-01)	8,00E-01 (4,81E-01)

Na Tabela 4.13, que apresenta a média e desvio padrão da acurácia do CMODESDE e NSGA-III para todas as instâncias, podemos perceber que o CMODESDE atingiu valores médios mais altos que o NSGA-III na maioria das instâncias, ficando atrás apenas em três delas.

A média e desvio padrão da estabilidade do CMODESDE e NSGA-III para todas as instâncias encontra-se na Tabela 4.14. Considerando que para a estabilidade quanto menor o valor melhor, o CMODESDE foi mais estável em 12 das 18 instâncias.

Tabela 4.13 – Média e desvio padrão das acurácias do CMODESDE e NSGA-III em cada uma das 18 instâncias

Instância	CMODESDE	NSGA-III
I1: sT10_dT10_sE5_dE1_SK4-5	8,47E-01 (1,77E-01)	8,72E-01 (1,61E-01)
I2: sT10_dT10_sE10_dE1_SK4-5	7,19E-01 (2,63E-01)	7,04E-01 (2,70E-01)
I3: sT10_dT10_sE15_dE1_SK4-5	7,72E-01 (2,12E-01)	7,45E-1 (2,32E-01)
I4: sT10_dT10_sE5_dE1_SK6-7	7,08E-01 (2,52E-01)	7,10E-01 (2,57E-01)
I5: sT10_dT10_sE10_dE1_SK6-7	7,15E-01 (2,75E-01)	7,11E-01 (2,86E-01)
I6: sT10_dT10_sE15_dE1_SK6-7	6,21E-01 (2,89E-01)	6,17E-01 (2,88E-01)
I7: sT20_dT10_sE5_dE1_SK4-5	7,37E-01 (2,60E-01)	7,36E-01 (2,60E-01)
I8: sT20_dT10_sE10_dE1_SK4-5	7,45E-01 (2,78E-01)	7,30E-01 (2,94E-01)
I9: sT20_dT10_sE15_dE1_SK4-5	7,35E-01 (2,85E-01)	7,10E-01 (3,04E-01)
I10: sT20_dT10_sE5_dE1_SK6-7	8,23E-01 (2,02E-01)	8,26E-01 (2,01E-01)
I11: sT20_dT10_sE10_dE1_SK6-7	7,31E-01 (2,80E-01)	7,13E-01 (2,94E-01)
I12: sT20_dT10_sE15_dE1_SK6-7	7,56E-01 (2,72E-01)	7,40E-01 (2,88E-01)
I13: sT30_dT10_sE5_dE1_SK4-5	6,69E-01 (2,51E-01)	6,66E-01 (2,58E-01)
I14: sT30_dT10_sE10_dE1_SK4-5	7,89E-01 (2,39E-01)	7,70E-01 (2,57E-01)
I15: sT30_dT10_sE15_dE1_SK4-5	7,95E-01 (2,70E-01)	7,69E-01 (2,90E-01)
I16: sT30_dT10_sE5_dE1_SK6-7	7,60E-01 (2,51E-01)	7,40E-01 (2,65E-01)
I17: sT30_dT10_sE10_dE1_SK6-7	7,34E-01 (2,90E-01)	7,19E-01 (3,05E-01)
I18: sT30_dT10_sE15_dE1_SK6-7	7,51E-01 (2,92E-01)	7,29E-01 (3,16E-01)

Tabela 4.14 – Média e desvio padrão das estabilidades do CMODESDE e NSGA-III em cada uma das 18 instâncias

Instância	CMODESDE	NSGA-III
I1: sT10_dT10_sE5_dE1_SK4-5	7,45E-02 (5,81E-01)	1,11E-01 (7,61E-01)
I2: sT10_dT10_sE10_dE1_SK4-5	4,13E-02 (2,09E-01)	5,28E-02 (2,94E-01)
I3: sT10_dT10_sE15_dE1_SK4-5	7,83E-02 (6,17E-01)	4,67E-02 (2,20E-01)
I4: sT10_dT10_sE5_dE1_SK6-7	7,28E-02 (4,55E-01)	6,18E-02 (4,00E-01)
I5: sT10_dT10_sE10_dE1_SK6-7	7,41E-02 (5,88E-01)	7,71E-02 (6,02E-01)
I6: sT10_dT10_sE15_dE1_SK6-7	5,84E-02 (4,85E-01)	6,38E-02 (4,98E-01)
I7: sT20_dT10_sE5_dE1_SK4-5	8,50E-02 (5,26E-01)	7,68E-02 (4,42E-01)
I8: sT20_dT10_sE10_dE1_SK4-5	4,48E-02 (3,57E-01)	7,31E-02 (5,42E-01)
I9: sT20_dT10_sE15_dE1_SK4-5	6,30E-02 (4,58E-01)	6,72E-02 (4,91E-01)
I10: sT20_dT10_sE5_dE1_SK6-7	7,95E-02 (4,45E-01)	6,80E-02 (3,33E-01)
I11: sT20_dT10_sE10_dE1_SK6-7	5,72E-02 (4,10E-01)	7,50E-02 (5,54E-01)
I12: sT20_dT10_sE15_dE1_SK6-7	4,98E-02 (3,96E-01)	7,04E-02 (5,38E-01)
I13: sT30_dT10_sE5_dE1_SK4-5	5,86E-02 (2,37E-01)	6,35E-02 (3,04E-01)
I14: sT30_dT10_sE10_dE1_SK4-5	5,69E-02 (4,56E-01)	7,34E-02 (5,60E-01)
I15: sT30_dT10_sE15_dE1_SK4-5	9,43E-02 (7,24E-01)	7,86E-02 (6,15E-01)
I16: sT30_dT10_sE5_dE1_SK6-7	7,01E-02 (4,45E-01)	7,61E-02 (4,73E-01)
I17: sT30_dT10_sE10_dE1_SK6-7	6,49E-02 (5,24E-01)	7,80E-02 (5,99E-01)
I18: sT30_dT10_sE15_dE1_SK6-7	6,46E-02 (4,83E-01)	6,04E-02 (4,46E-01)

Analisando em mais detalhes a instância I8, a Figura 4.14 apresenta os valores dos hipervolumes. Podemos perceber que o CMODESDE começa um pouco melhor que o NSGA-III. Porém, à medida que os eventos vão ocorrendo, o NSGA-III começa a obter melhores resultados, principalmente entre os pontos de reescalonamento 50 e 60. Ao realizar o teste de Wilcoxon, obteve um valor-*p* de 4,935780E-01, afirmando que não existe diferença estatística. Esse teste confirma o que é mostrado na Tabela 4.15, que mostra a comparação pareada dos valores-*p* entre os algoritmos.

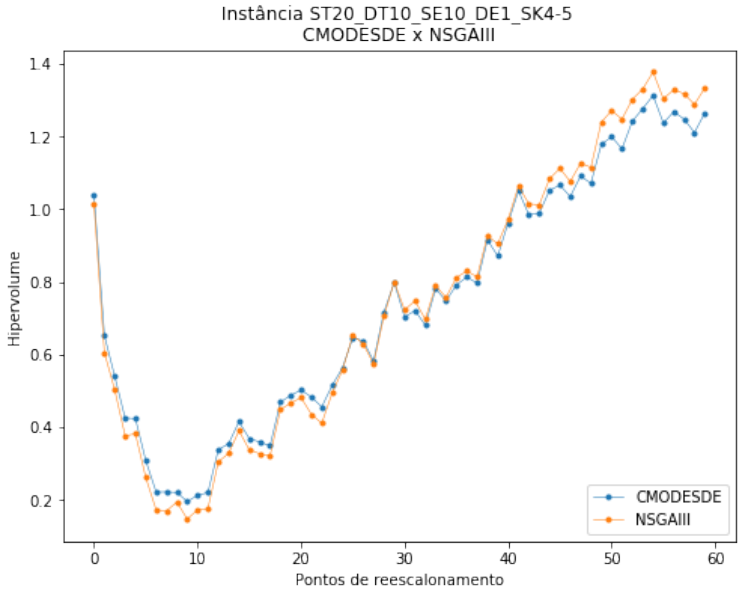

Figura 4.14 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I8

Tabela 4.15 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos CMODESDE e NSGA-III para a instância I8

	CMODESDE	NSGA-III
CMODESDE	-1,00E+00	9,25E-01
NSGA-III	9,25E-01	-1,00E+00

Na Figura 4.15, podemos perceber que a acurácia tem um comportamento semelhante ao hipervolume para a instância I8. E assim como no hipervolume, o teste de Wilcoxon para a acurácia, afirma que não existe diferença estatística, com um valor-*p* de 8,032964E-01. No gráfico que representa a estabilidade (Figura 4.16), podemos perceber

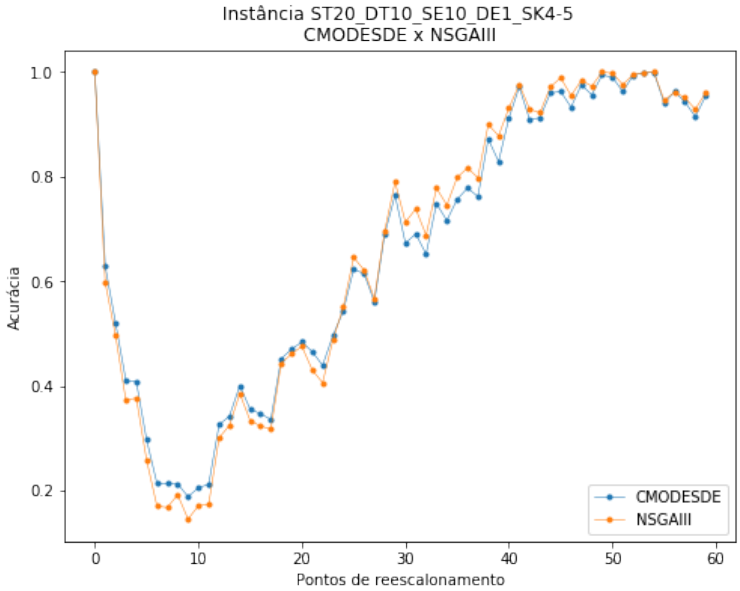

Figura 4.15 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I8

que os algoritmos são relativamente estáveis em relação aos pontos de reescalonamento, tendo poucos picos. O valor-*p* obtido pelo teste de Wilcoxon para a estabilidade foi de 7,664264E-01, confirmando que não existe diferença estatística.

Na instância I15 o CMODESDE apresentou melhores resultados em duas das três métricas. Na Figura 4.17, podemos perceber que o CMODESDE obteve melhores valores de hipervolume que o NSGA-III, ficando mais clara a diferença a partir do ponto de reescalonamento 30. Para a acurácia, Figura 4.18, o início é semelhante ao hipervolume, porém com uma diferença menor, e quase se equiparando a partir do ponto de reescalonamento 50. Considerando a estabilidade, o NSGA-III obteve menos picos, como pode ser visto na Figura 4.19. O pós-teste dos testes estatísticos realizando uma comparação pareada entre o CMODESDE e o NSGA-III, obteve um valor-*p* de 1,19E-01, confirmando que não existe diferença estatística entre eles.

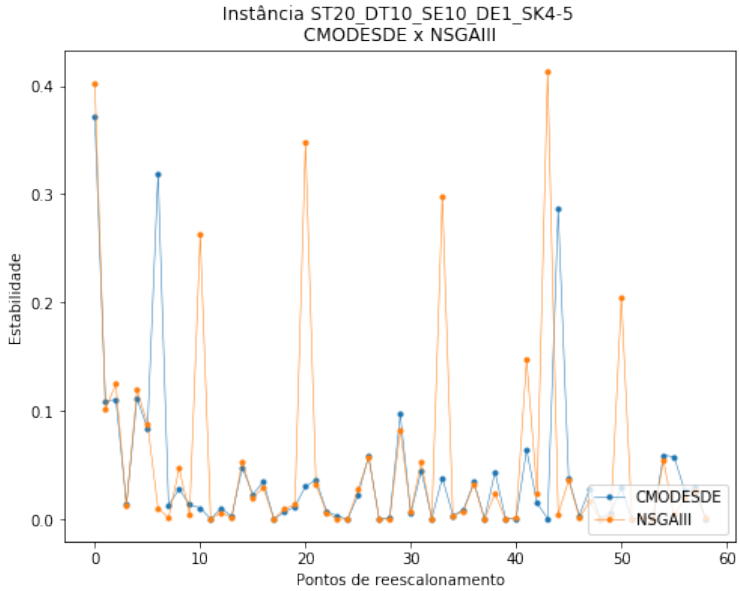

Figura 4.16 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I8

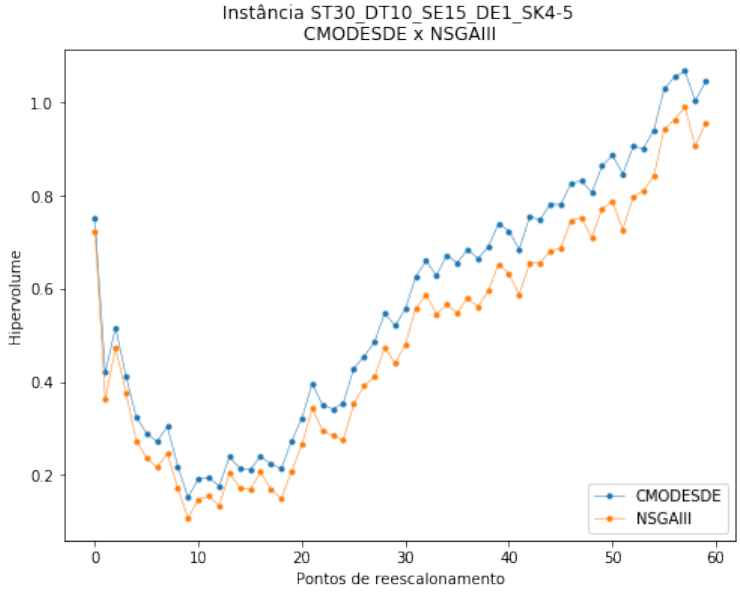

Figura 4.17 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I15

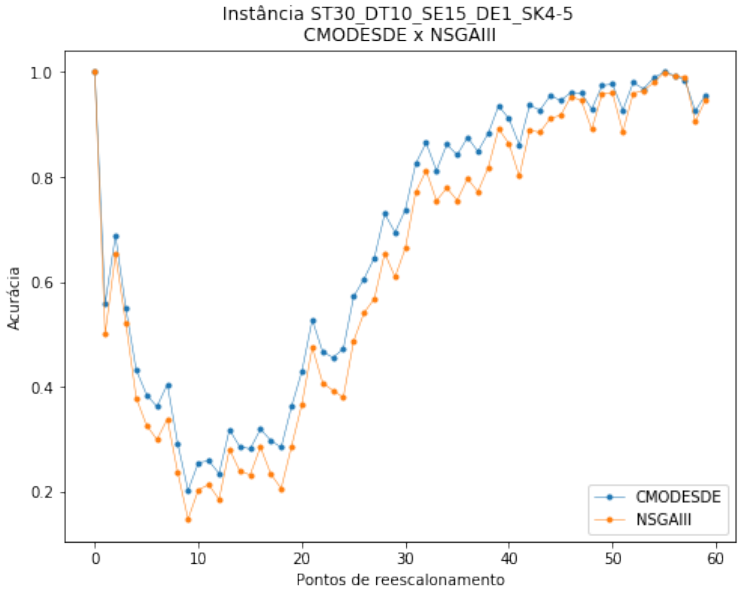

Figura 4.18 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I15

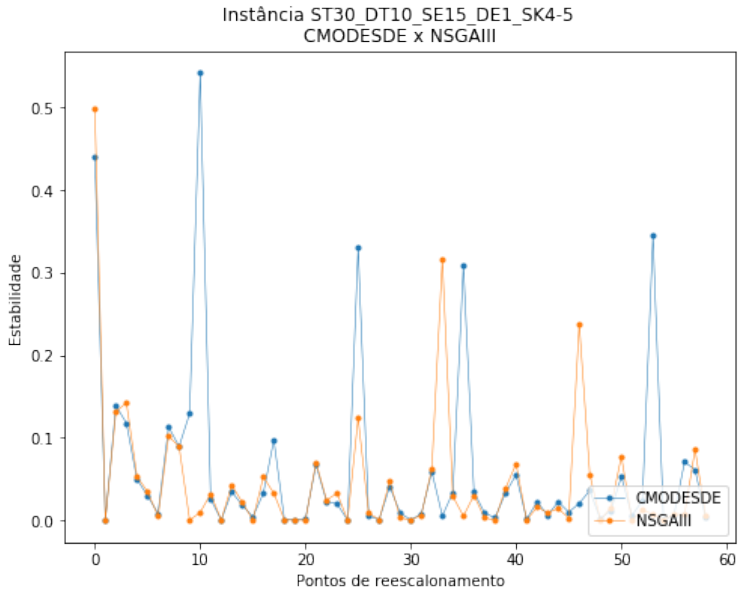

Figura 4.19 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I15

De modo geral, o CMODESDE apresentou melhores resultados em comparação

com NSGA-III em todas as métricas. Apesar disso, os testes de comparação realizados indicaram que não existe diferença significativa entre eles em nenhuma das instâncias. Outra informação obtida durante os experimentos, é que o tempo de execução do NSGA-III pode chegar a ser até seis vezes mais lento que o CMODESDE, por exemplo, na instância que contém mais informações, o CMODESDE tem um tempo de execução médio de cinco minutos e trinta segundos, enquanto o NSGA-III tem um tempo de execução de mais de trinta minutos. Uma situação que pode-se destacar em relação ao NSGA-III é que para a instância I10, ele obteve melhor resultados para todas as métricas, porém esse bom desempenho não se repete em nenhum caso e nenhum padrão pode ser identificado. Para melhor entender essa situação, é necessário uma análise detalhada da execução do NSGA-III nessa instância.

4.2.4 Influência das estratégias dinâmicas

O experimento relativo a QP4 tem como objetivo investigar a influência das técnicas dinâmicas para o desempenho dos algoritmos. Para isso foram executadas variantes do CMODESDE com diferentes combinações de técnicas dinâmicas de acordo com a seção 3.2.1.1, são elas: CMODESDEDynamic, usando informação histórica; CMODESDEExternalDynamic, usando um arquivo externo contendo as soluções escolhidas, em que uma delas é escolhida para gerar o vetor mutante na execução do DE para o arquivo; CMODESDEExternalReDynamic, semelhante ao anterior, porém fazendo a reavaliação da solução; CMODESDERepairDynamic, que utiliza ajustes proativos da solução a cada ponto de reescalonamento; CMODESDEFullDynamic, que mistura as técnicas utilizadas pelo CMODESDEDynamic e CMODESDEExternalDynamic; e CMODESDEFullReDynamic, que usa as técnicas do CMODESDEDynamic e CMODESDEExternalReDynamic. A Tabela 4.16 apresenta o identificador que podem ser utilizados em alguns casos no lugar do nome do algoritmo para facilitar a visualização das tabelas. Será discutido no texto uma instância para cada variação da quantidade de tarefas iniciais do projeto, ou seja, uma que começa com 10, depois com 20 e por último com 30 tarefas iniciais.

As Tabelas 4.17 e 4.18, apresentam a média e o desvio padrão do hipervolume dos algoritmos dinâmicos para as instâncias I1 a I10 e I11 a I18, respectivamente. O algoritmo que se saiu melhor considerando o hipervolume foi o CMODESDERepairDynamic, obtendo a maior média em 13 das 18 instâncias.

Tabela 4.16 – Identificadores dos algoritmos dinâmcos

Algoritmo	Identificador
CMODESDEDynamic	CMODESDE-D
CMODESDEExternalDynamic	CMODESDE-ED
CMODESDEExternalReDynamic	CMODESDE-ERD
CMODESDEFullDynamic	CMODESDE-FD
CMODESDEFullReDynamic	CMODESDE-FRD
CMODESDERepairDynamic	CMODESDE-RD
NSGAIIIDynamic	NSGAIII-D

Nas Tabelas 4.19 e 4.20, são exibidas as médias e desvio padrão da acurácia dos algoritmos dinâmicos para as instâncias I1 a I10 e I11 a I18, respectivamente. Nela, nota-se que o algoritmo que obteve melhor média de acurácia na maioria das instâncias também foi o CMODESDERepairDynamic, possuindo maiores valores em 14 das 18 instâncias.

A média e desvio padrão da estabilidade para os algoritmos dinâmicos para as instâncias I1 a I10 e I11 a I18 encontram-se, respectivamente, nas Tabelas 4.21 e 4.22. O CMODESDEExternalReDynamic foi o algoritmo mais estável em 9 das 18 instâncias e o CMODESDERepairDynamic não foi o melhor em nenhuma das instâncias.

Tabela 4.17 – Média e desvio padrão dos hipervolumes dos algoritmos com o uso de técnicas dinâmicas nas instâncias I1 a I10

Instância	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
I1	1,32E+00 (2,64E-01)	1,25E+00 (2,66E-01)	1,24E+00 (2,69E-01)	1,32E+00 (2,65E-01)	1,32E+00 (2,64E-01)	1,35E+00 (2,09E-01)	1,35E+00 (2,03E-01)
I2	1,26E+00 (2,73E-01)	1,21E+00 (2,58E-01)	1,21E+00 (2,56E-01)	1,25E+00 (2,68E-01)	1,25E+00 (2,80E-01)	1,31E+00 (2,28E-01)	1,23E+00 (2,60E-01)
I3	1,30E+00 (2,18E-01)	1,29E+00 (1,98E-01)	1,29E+00 (1,98E-01)	1,31E+00 (2,27E-01)	1,30E+00 (2,15E-01)	1,42E+00 (1,45E-01)	1,41E+00 (1,71E-01)
I4	1,27E+00 (2,80E-01)	1,20E+00 (2,88E-01)	1,18E+00 (3,07E-01)	1,27E+00 (2,80E-01)	1,26E+00 (2,82E-01)	1,26E+00 (2,82E-01)	1,24E+00 (2,83E-01)
I5	1,28E+00 (2,72E-01)	1,26E+00 (2,60E-01)	1,26E+00 (2,76E-01)	1,29E+00 (2,75E-01)	1,28E+00 (2,83E-01)	1,37E+00 (1,88E-01)	1,30E+00 (2,58E-01)
I6	1,24E+00 (2,55E-01)	1,21E+00 (2,40E-01)	1,20E+00 (2,50E-01)	1,23E+00 (2,51E-01)	1,22E+00 (2,57E-01)	1,29E+00 (2,22E-01)	1,22E+00 (2,64E-01)
I7	1,25E+00 (2,73E-01)	1,14E+00 (3,30E-01)	1,13E+00 (3,35E-01)	1,24E+00 (2,73E-01)	1,24E+00 (2,78E-01)	1,18E+00 (2,41E-01)	1,12E+00 (2,93E-01)
I8	1,29E+00 (3,03E-01)	1,24E+00 (2,90E-01)	1,23E+00 (2,91E-01)	1,29E+00 (3,00E-01)	1,27E+00 (3,11E-01)	1,38E+00 (1,95E-01)	1,31E+00 (2,60E-01)
I9	1,24E+00 (3,01E-01)	1,20E+00 (2,94E-01)	1,19E+00 (2,90E-01)	1,24E+00 (2,99E-01)	1,22E+00 (3,11E-01)	1,30E+00 (2,48E-01)	1,26E+00 (2,47E-01)
I10	1,30E+00 (2,70E-01)	1,22E+00 (3,08E-01)	1,22E+00 (3,04E-01)	1,29E+00 (2,76E-01)	1,29E+00 (2,71E-01)	1,28E+00 (2,33E-01)	1,24E+00 (2,68E-01)

Tabela 4.18 – Média e desvio padrão dos hipervolumes dos algoritmos com o uso de técnicas dinâmicas nas instâncias I11 a I18

Instância	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
I11	1,28E+00 (2,87E-01)	1,18E+00 (3,14E-01)	1,17E+00 (3,14E-01)	1,26E+00 (2,87E-01)	1,25E+00 (2,97E-01)	1,30E+00 (2,35E-01)	1,24E+00 (2,76E-01)
I12	1,23E+00 (2,80E-01)	1,20E+00 (2,92E-01)	1,19E+00 (2,89E-01)	1,24E+00 (2,72E-01)	1,21E+00 (2,98E-01)	1,26E+00 (2,54E-01)	1,20E+00 (2,93E-01)
I13	1,14E+00 (2,68E-01)	1,03E+00 (2,81E-01)	1,04E+00 (2,88E-01)	1,14E+00 (2,74E-01)	1,11E+00 (2,77E-01)	1,19E+00 (1,80E-01)	1,16E+00 (2,01E-01)
I14	1,32E+00 (2,36E-01)	1,26E+00 (2,39E-01)	1,27E+00 (2,40E-01)	1,32E+00 (2,31E-01)	1,31E+00 (2,44E-01)	1,34E+00 (2,18E-01)	1,30E+00 (2,19E-01)
I15	1,24E+00 (3,07E-01)	1,20E+00 (2,94E-01)	1,20E+00 (3,00E-01)	1,25E+00 (3,01E-01)	1,24E+00 (3,06E-01)	1,30E+00 (2,49E-01)	1,28E+00 (2,68E-01)
I16	1,21E+00 (2,67E-01)	1,11E+00 (2,79E-01)	1,11E+00 (2,74E-01)	1,21E+00 (2,59E-01)	1,17E+00 (2,68E-01)	1,18E+00 (1,88E-01)	1,10E+00 (2,22E-01)
I17	1,26E+00 (2,80E-01)	1,17E+00 (3,12E-01)	1,18E+00 (3,14E-01)	1,25E+00 (2,84E-01)	1,23E+00 (2,93E-01)	1,24E+00 (2,45E-01)	1,20E+00 (2,90E-01)
I18	1,24E+00 (3,14E-01)	1,22E+00 (3,03E-01)	1,23E+00 (3,03E-01)	1,25E+00 (3,04E-01)	1,23E+00 (3,21E-01)	1,27E+00 (2,97E-01)	1,24E+00 (3,09E-01)

Tabela 4.19 – Média e desvio padrão das acurácias dos algoritmos com o uso de técnicas dinâmicas nas instâncias I1 a I10

Instância	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
I1	9,28E-01 (1,27E-01)	9,03E-01 (1,27E-01)	9,00E-01 (1,30E-01)	9,27E-01 (1,28E-01)	9,28E-01 (1,27E-01)	9,52E-01 (7,51E-02)	9,58E-01 (7,04E-02)
I2	8,92E-01 (1,52E-01)	8,69E-01 (1,47E-01)	8,71E-01 (1,44E-01)	8,91E-01 (1,51E-01)	8,86E-01 (1,57E-01)	9,25E-01 (1,17E-01)	8,85E-01 (1,45E-01)
I3	9,51E-01 (9,80E-02)	9,53E-01 (8,05E-02)	9,53E-01 (8,18E-02)	9,51E-01 (1,04E-01)	9,53E-01 (9,23E-02)	9,87E-01 (2,83E-02)	9,84E-01 (3,63E-02)
I4	9,01E-01 (1,44E-01)	8,73E-01 (1,54E-01)	8,60E-01 (1,68E-01)	9,04E-01 (1,42E-01)	8,97E-01 (1,47E-01)	9,01E-01 (1,44E-01)	8,93E-01 (1,47E-01)
I5	9,07E-01 (1,48E-01)	9,01E-01 (1,41E-01)	8,95E-01 (1,53E-01)	9,05E-01 (1,50E-01)	9,01E-01 (1,56E-01)	9,55E-01 (7,69E-02)	9,14E-01 (1,34E-01)
I6	8,94E-01 (1,31E-01)	8,85E-01 (1,21E-01)	8,80E-01 (1,28E-01)	8,95E-01 (1,29E-01)	8,86E-01 (1,36E-01)	9,23E-01 (9,82E-02)	8,90E-01 (1,29E-01)
I7	9,19E-01 (1,06E-01)	8,81E-01 (1,55E-01)	8,76E-01 (1,60E-01)	9,18E-01 (1,11E-01)	9,15E-01 (1,13E-01)	9,29E-01 (8,77E-02)	8,90E-01 (1,25E-01)
I8	9,32E-01 (1,34E-01)	9,26E-01 (1,18E-01)	9,25E-01 (1,19E-01)	9,34E-01 (1,34E-01)	9,26E-01 (1,41E-01)	9,81E-01 (3,93E-02)	9,59E-01 (8,89E-02)
I9	9,30E-01 (1,31E-01)	9,19E-01 (1,20E-01)	9,20E-01 (1,18E-01)	9,31E-01 (1,32E-01)	9,23E-01 (1,42E-01)	9,63E-01 (7,22E-02)	9,62E-01 (6,32E-02)
I10	9,67E-01 (8,20E-02)	9,42E-01 (1,00E-01)	9,44E-01 (9,71E-02)	9,63E-01 (8,62E-02)	9,63E-01 (8,15E-02)	9,71E-01 (5,50E-02)	9,61E-01 (6,59E-02)

Tabela 4.20 – Média e desvio padrão das acurácias dos algoritmos com o uso de técnicas dinâmicas nas instâncias I11 a I18

Instância	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
I11	9,45E-01 (1,04E-01)	9,09E-01 (1,26E-01)	9,10E-01 (1,25E-01)	9,42E-01 (1,04E-01)	9,35E-01 (1,11E-01)	9,67E-01 (4,74E-02)	9,50E-01 (7,80E-02)
I12	9,39E-01 (1,12E-01)	9,23E-01 (1,15E-01)	9,24E-01 (1,14E-01)	9,46E-01 (1,06E-01)	9,25E-01 (1,26E-01)	9,55E-01 (7,61E-02)	9,32E-01 (1,02E-01)
I13	9,09E-01 (1,29E-01)	8,76E-01 (1,58E-01)	8,77E-01 (1,56E-01)	9,08E-01 (1,32E-01)	8,90E-01 (1,44E-01)	9,68E-01 (4,74E-02)	9,53E-01 (6,21E-02)
I14	9,61E-01 (8,59E-02)	9,51E-01 (8,54E-02)	9,52E-01 (8,35E-02)	9,63E-01 (8,31E-02)	9,58E-01 (9,28E-02)	9,70E-01 (5,90E-02)	9,71E-01 (5,07E-02)
I15	9,52E-01 (9,78E-02)	9,57E-01 (7,76E-02)	9,55E-01 (7,98E-02)	9,54E-01 (9,80E-02)	9,52E-01 (9,66E-02)	9,81E-01 (4,23E-02)	9,83E-01 (3,70E-02)
I16	9,39E-01 (1,04E-01)	9,13E-01 (1,18E-01)	9,16E-01 (1,14E-01)	9,42E-01 (9,70E-02)	9,21E-01 (1,13E-01)	9,61E-01 (4,76E-02)	9,31E-01 (7,85E-02)
I17	9,57E-01 (9,50E-02)	9,28E-01 (1,23E-01)	9,26E-01 (1,26E-01)	9,53E-01 (9,83E-02)	9,46E-01 (1,09E-01)	9,64E-01 (7,21E-02)	9,49E-01 (9,32E-02)
I18	9,32E-01 (1,31E-01)	9,34E-01 (1,15E-01)	9,37E-01 (1,14E-01)	9,38E-01 (1,24E-01)	9,27E-01 (1,37E-01)	9,47E-01 (1,08E-01)	9,46E-01 (1,01E-01)

Tabela 4.21 – Média e desvio padrão das estabilidades dos algoritmos com o uso de técnicas dinâmicas nas instâncias I1 a I10

Instância	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
I1	1,23E-01 (8,04E-01)	1,16E-01 (8,02E-01)	1,02E-01 (6,96E-01)	1,54E-01 (9,30E-01)	1,41E-01 (9,27E-01)	1,10E-01 (7,64E-01)	1,62E-01 (9,96E-01)
I2	1,14E-01 (7,69E-01)	1,06E-01 (7,84E-01)	9,83E-02 (7,22E-01)	1,45E-01 (9,33E-01)	1,23E-01 (8,08E-01)	1,63E-01 (9,96E-01)	1,54E-01 (9,52E-01)
I3	1,44E-01 (8,87E-01)	1,43E-01 (9,09E-01)	1,05E-01 (7,46E-01)	1,57E-01 (9,40E-01)	1,43E-01 (9,34E-01)	2,69E-01 (1,31E+00)	4,63E-01 (1,58E+00)
I4	1,23E-01 (8,22E-01)	8,28E-02 (6,53E-01)	6,75E-02 (5,47E-01)	5,71E-02 (4,04E-01)	9,41E-02 (6,86E-01)	1,26E-01 (8,73E-01)	1,19E-01 (8,02E-01)
I5	1,12E-01 (7,80E-01)	1,17E-01 (8,49E-01)	1,41E-01 (9,07E-01)	1,44E-01 (9,32E-01)	1,71E-01 (1,01E+00)	1,71E-01 (1,02E+00)	1,87E-01 (1,08E+00)
I6	1,64E-01 (9,68E-01)	1,71E-01 (1,00E+00)	1,42E-01 (9,31E-01)	2,04E-01 (1,08E+00)	1,43E-01 (8,74E-01)	1,80E-01 (9,91E-01)	1,77E-01 (1,05E+00)
I7	1,12E-01 (6,79E-01)	1,19E-01 (8,24E-01)	6,73E-02 (5,17E-01)	1,36E-01 (8,75E-01)	1,61E-01 (9,34E-01)	8,97E-02 (6,82E-01)	1,28E-01 (8,68E-01)
I8	2,41E-01 (1,21E+00)	1,58E-01 (9,73E-01)	1,81E-01 (1,05E+00)	2,14E-01 (1,15E+00)	1,96E-01 (1,09E+00)	2,20E-01 (1,18E+00)	2,40E-01 (1,22E+00)
I9	2,43E-01 (1,20E+00)	1,62E-01 (9,93E-01)	1,34E-01 (8,67E-01)	2,55E-01 (1,22E+00)	2,28E-01 (1,16E+00)	2,66E-01 (1,29E+00)	2,67E-01 (1,28E+00)
I10	1,00E-01 (7,59E-01)	1,06E-01 (7,38E-01)	1,24E-01 (8,51E-01)	8,44E-02 (6,84E-01)	7,78E-02 (6,02E-01)	1,08E-01 (7,63E-01)	1,17E-01 (8,30E-01)

Tabela 4.22 – Média e desvio padrão das estabilidades dos algoritmos com o uso de técnicas dinâmicas nas instâncias I11 a I18

Instância	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
I11	1,70E-01 (9,84E-01)	1,28E-01 (8,43E-01)	1,47E-01 (9,55E-01)	1,69E-01 (9,89E-01)	1,71E-01 (1,01E+00)	1,63E-01 (9,91E-01)	1,76E-01 (1,03E+00)
I12	1,64E-01 (9,52E-01)	1,94E-01 (1,08E+00)	1,71E-01 (1,04E+00)	1,90E-01 (1,05E+00)	1,79E-01 (1,03E+00)	2,40E-01 (1,21E+00)	1,91E-01 (1,10E+00)
I13	9,53E-02 (6,52E-01)	5,79E-02 (5,08E-01)	9,38E-02 (7,37E-01)	1,16E-01 (8,00E-01)	8,46E-02 (5,63E-01)	1,25E-01 (8,73E-01)	9,31E-02 (7,54E-01)
I14	2,39E-01 (1,23E+00)	1,66E-01 (1,00E+00)	1,59E-01 (9,64E-01)	1,54E-01 (9,49E-01)	2,12E-01 (1,13E+00)	1,96E-01 (1,12E+00)	2,31E-01 (1,20E+00)
I15	1,79E-01 (1,04E+00)	1,63E-01 (9,97E-01)	1,59E-01 (9,68E-01)	2,22E-01 (1,15E+00)	2,12E-01 (1,12E+00)	2,35E-01 (1,23E+00)	2,88E-01 (1,39E+00)
I16	1,33E-01 (8,78E-01)	1,03E-01 (7,57E-01)	6,05E-02 (5,52E-01)	9,86E-02 (7,66E-01)	8,21E-02 (6,06E-01)	1,09E-01 (8,08E-01)	7,61E-02 (6,47E-01)
I17	2,03E-01 (1,12E+00)	1,88E-01 (1,08E+00)	1,71E-01 (9,98E-01)	2,36E-01 (1,19E+00)	1,95E-01 (1,12E+00)	1,85E-01 (1,12E+00)	1,51E-01 (9,50E-01)
I18	1,98E-01 (1,10E+00)	2,05E-01 (1,16E+00)	1,63E-01 (9,83E-01)	1,83E-01 (1,05E+00)	1,87E-01 (1,03E+00)	2,99E-01 (1,37E+00)	2,69E-01 (1,30E+00)

Na Figura 4.20 não é possível visualizar com clareza qual o algoritmo que se sobressai, considerando o hipervolume, quando aplicado a instância I6. Porém, podemos observar que o NSGAIIIDynamic está entre os piores e que o CMODESDERepairDynamic começa um pouco melhor que os outros, entre os pontos de reescalonamento 0 e 20. Consultando as médias dos hipervolumes na Tabela 4.17, vemos que o algoritmo que obteve a melhor média foi o CMODESDERepairDynamic com 1,29. Após o teste de Friedman, foi identificado que os dados possuem distribuições diferentes, com um valor-*p* de 2,796342E-24.

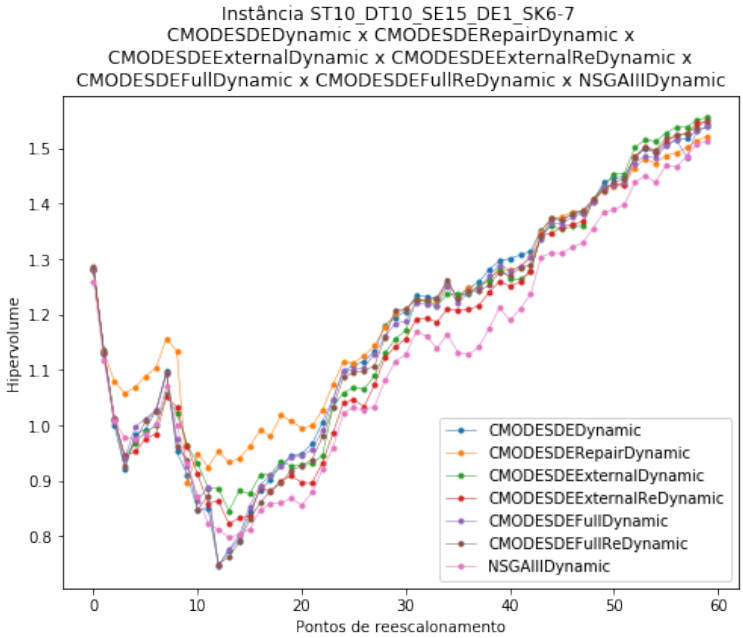

Figura 4.20 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I6

Considerando a acurácia para a mesma instância, Figura 4.21, podemos perceber que ela é semelhante ao hipervolume, com destaque para o CMODESDERepairDynamic entre os pontos 0 e 20. Verificando as médias das acurácias na Tabela 4.19, vemos que o algoritmo que obteve a melhor acurácia foi o CMODESDERepairDynamic com 0,923. O teste de Friedman para a acurácia também indicou que os dados possuem distribuições diferentes, com um valor-*p* de 1,413988E-19.

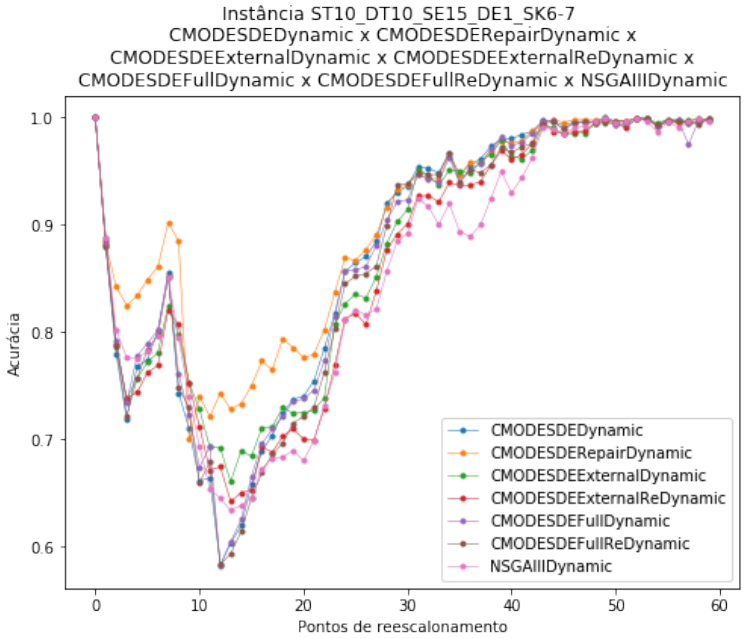

Figura 4.21 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I6

Em relação à estabilidade, Figura 4.22, não é possível identificar com clareza qual o algoritmo mais estável. Porém, consultando a Tabela 4.21, vemos que o algoritmo que teve a melhor média de estabilidade foi o CMODESDEExternalReDynamic com 0,142. O teste estatístico de Friedman para a estabilidade obteve um valor-*p* de 6,706295E-02, indicando que não tem diferença estatística em relação à estabilidade.

Realizando uma comparação pareada em relação ao hipervolume (Tabela 4.23) indica que, estatisticamente, há diferença entre os algoritmos. Na Figura 4.23, fica mais fácil de identificar que os algoritmos que possuem uma diferença estatística são o CMODESDERepairDynamic e o NSGAIIIDynamic.

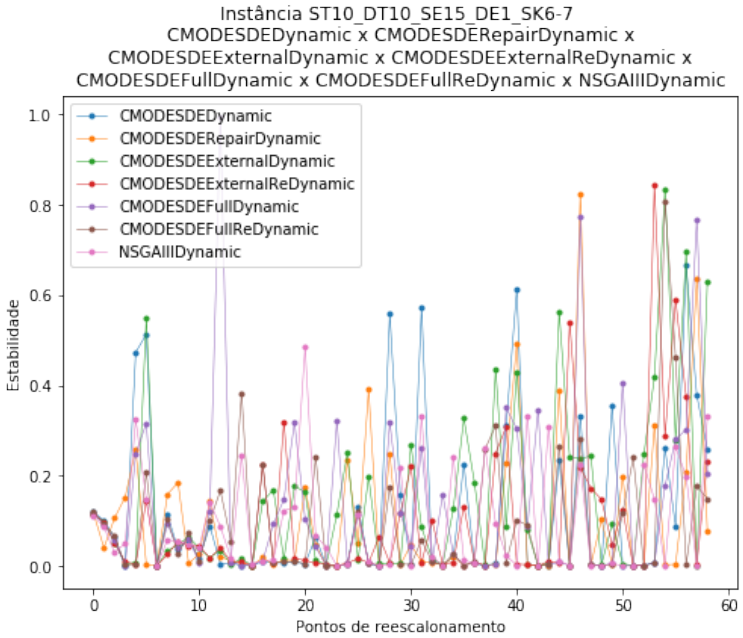

Figura 4.22 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I6

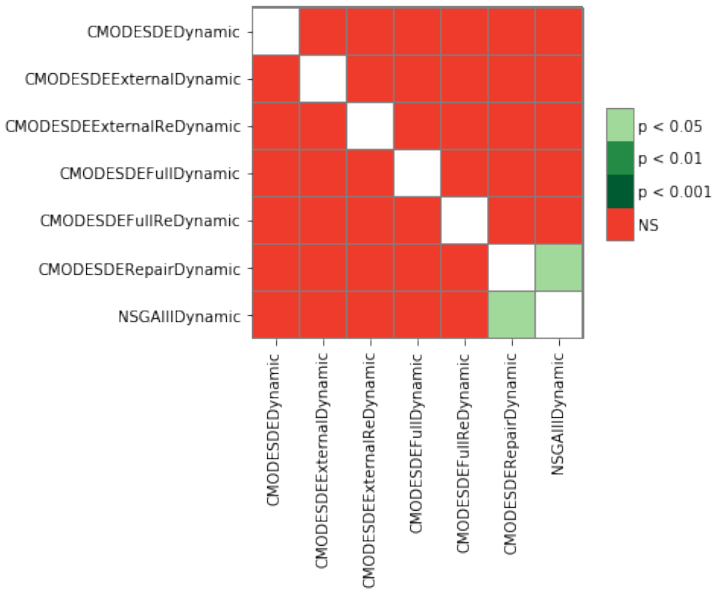

Figura 4.23 – Comparativo da significância indicada pelo valor-*p* entre todos os algoritmos dinâmicos para a instância I6

Tabela 4.23 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos dinâmicos para a instância I6

	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
CMODESDE-D	-1,00E+00	9,44E-01	5,95E-01	9,22E-01	8,74E-01	4,91E-01	1,90E-01
CMODESDE-ED	9,44E-01	-1,00E+00	6,45E-01	9,78E-01	9,30E-01	4,47E-01	2,15E-01
CMODESDE-ERD	5,95E-01	6,45E-01	-1,00E+00	6,65E-01	7,09E-01	2,22E-01	4,36E-01
CMODESDE-FD	9,22E-01	9,78E-01	6,65E-01	-1,00E+00	9,52E-01	4,31E-01	2,26E-01
CMODESDE-FRD	8,74E-01	9,30E-01	7,09E-01	9,52E-01	-1,00E+00	3,97E-01	2,49E-01
CMODESDE-RD	4,91E-01	4,47E-01	2,22E-01	4,31E-01	3,97E-01	-1,00E+00	4,60E-02
NSGAIII-D	1,90E-01	2,15E-01	4,36E-01	2,26E-01	2,49E-01	4,60E-02	-1,00E+00

Na Figura 4.24 também não é possível visualizar com clareza qual o algoritmo que se sobressai, considerando o hipervolume, quando aplicado a instância I7, se é o CMODESDEFullDynamic, CMODESDEFullReDynamic ou CMODESDEDynamic. Verificando as médias dos hipervolumes, temos que o CMODESDEDynamic apresentou a maior média com 1,25. O teste estatístico de Friedman para o hipervolume, indicou que há diferença significativa entre os algoritmos com um valor-*p* de 3,363345E-45.

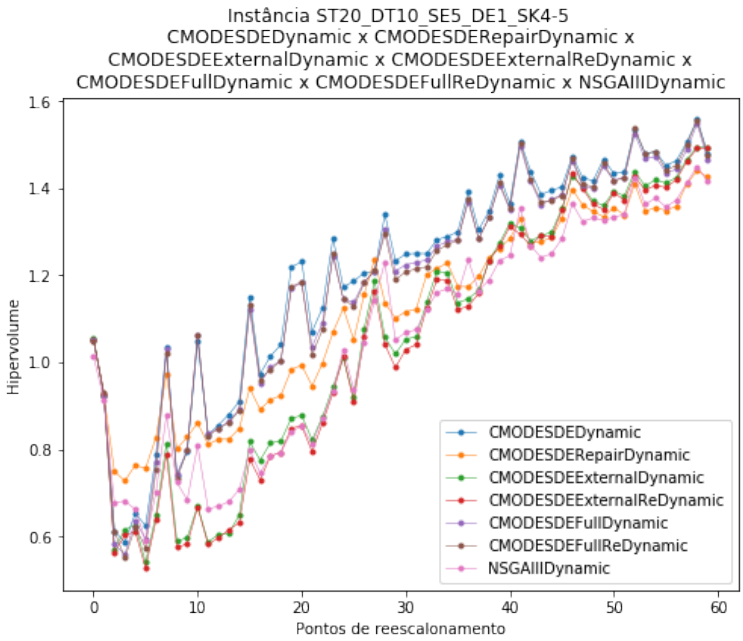

Figura 4.24 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I7

Para a acurácia, fica mais difícil ainda identificar pelo gráfico da Figura 4.25, o algoritmo que se sobressai para essa instância. Verificando as médias das acurácias, temos que o CMODESDERepairDynamic obteve a maior média com um valor de 0,929. O teste estatístico de Friedman para a acurácia, indicou que há diferença significativa entre os algoritmos com um valor-*p* de 9,241270E-05.

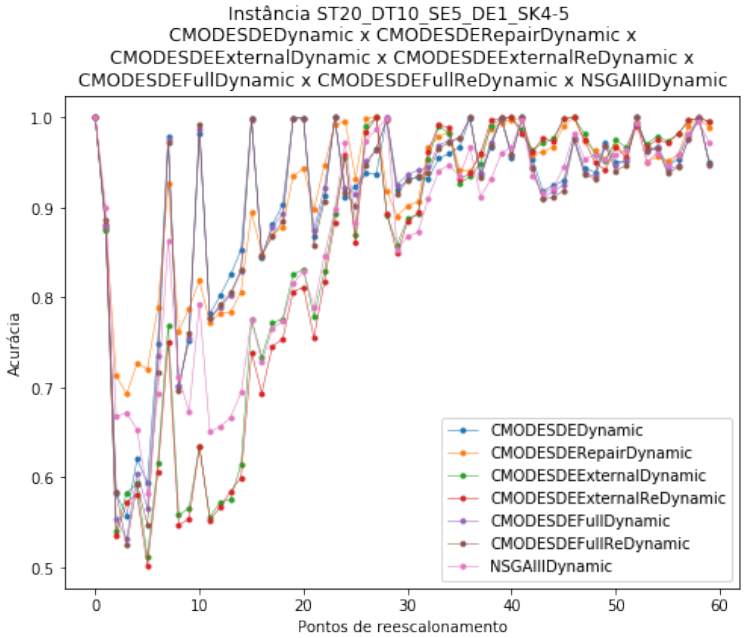

Figura 4.25 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I7

Considerando a estabilidade, Figura 4.26, não fica claro na visualização qual algoritmo foi melhor, uma vez que existem muitos picos. Verificando a tabela das médias das estabilidades, temos que o algoritmo mais estável foi o CMODESDEExternalReDynamic com média de 0,0673. Entretanto, o teste estatístico para a estabilidade indicou que não existe uma diferença significativa entre eles, obtendo um valor-*p* de 1,614427E-01.

Pela comparação pareada considerando o hipervolume, Tabela 4.24, temos que vários algoritmos possuem diferenças significativas. Podemos perceber mais claramente os algoritmos que possuem diferença na Figura 4.27. Nela, podemos notar que os algoritmos que tiveram diferença estatística com o NSGA-III e o CMODESDEExternalReDynamic, tiveram um nível de confiança de mais de 99% e que o único que apresentou diferença estatística com o CMODESDERepairDynamic foi o CMODESDEDynamic.

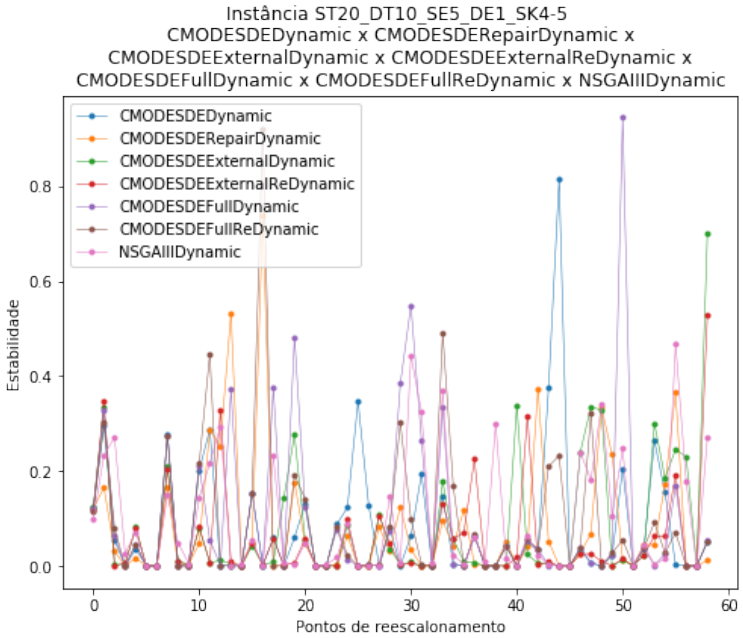

Figura 4.26 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I7

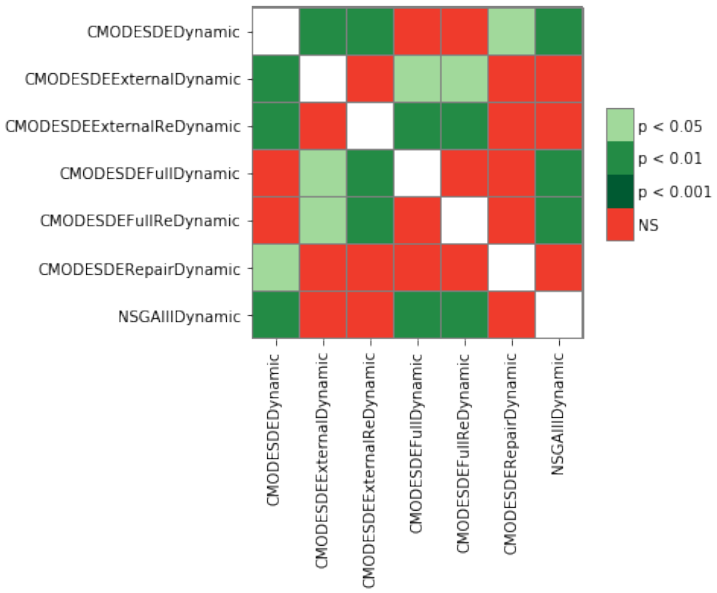

Figura 4.27 – Comparativo da significância indicada pelo valor-*p* entre todos os algoritmos dinâmicos para a instância I7

Tabela 4.24 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos dinâmicos para a instância I7

	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
CMODESDE-D	-1,00E+00	4,47E-03	1,80E-03	6,47E-01	6,51E-01	2,62E-02	1,59E-03
CMODESDE-ED	4,47E-03	-1,00E+00	7,76E-01	1,68E-02	1,66E-02	5,31E-01	7,48E-01
CMODESDE-ERD	1,80E-03	7,76E-01	-1,00E+00	7,57E-03	7,45E-03	3,63E-01	9,71E-01
CMODESDE-FD	6,47E-01	1,68E-02	7,57E-03	-1,00E+00	9,96E-01	7,69E-02	6,79E-03
CMODESDE-FRD	6,51E-01	1,66E-02	7,45E-03	9,96E-01	-1,00E+00	7,60E-02	6,68E-03
CMODESDE-RD	2,62E-02	5,31E-01	3,63E-01	7,69E-02	7,60E-02	-1,00E+00	3,44E-01
NSGAIII-D	1,59E-03	7,48E-01	9,71E-01	6,79E-03	6,68E-03	3,44E-01	-1,00E+00

Na Figura 4.28, podemos perceber que a partir do ponto de reescalonamento 25 o algoritmo CMODESDEDynamic está acima dos demais quando aplicado a instância I17. Verificando as médias do hipervolume, confirma-se o que foi observado na figura, o CMODESDEDynamic apresentou a maior média com valor de 1,26. O teste estatístico de Friedman indicou que existe diferenças significativas entre os algoritmos, considerando o hipervolume, obtendo um valor-*p* de 3,261378E-50.

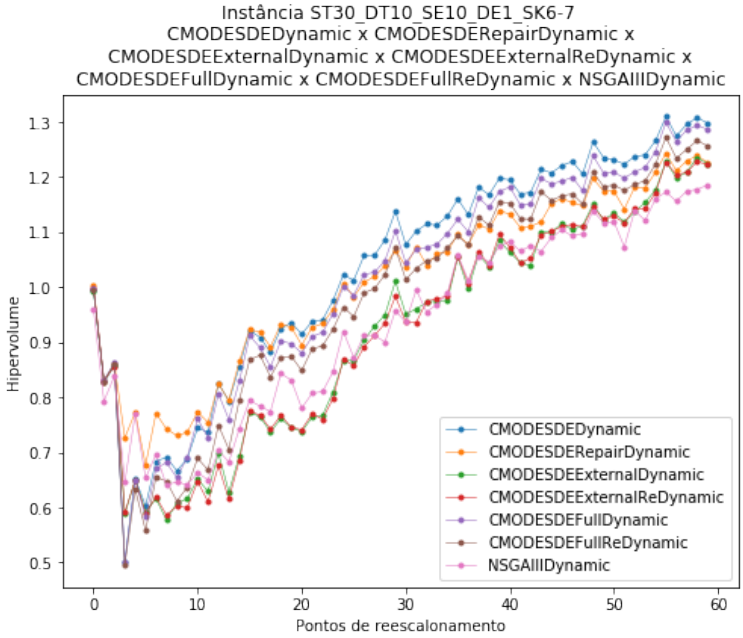

Figura 4.28 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I17

Considerando a acurácia, Figura 4.29, mais uma vez, não se pode visualizar claramente qual algoritmo é melhor para esta instância. Consultando as médias da mesma, temos que a maior média foi 0,964 obtida pelo algoritmo CMODESDERepairDynamic. O teste estatístico para a acurácia apresentou um valor-*p* de 2,558246E-15, indicando que os dados possuem distribuições diferentes.

Novamente, na Figura 4.30, não fica claro qual o algoritmo mais estável quando aplicado a instância I17. Observando as médias da estabilidade, temos que o algoritmo com a melhor média é o NSGAIIIDynamic com valor de 0,151. Assim como aconteceu nas outras instâncias, o teste estatístico mostrou que não existe diferença significativa

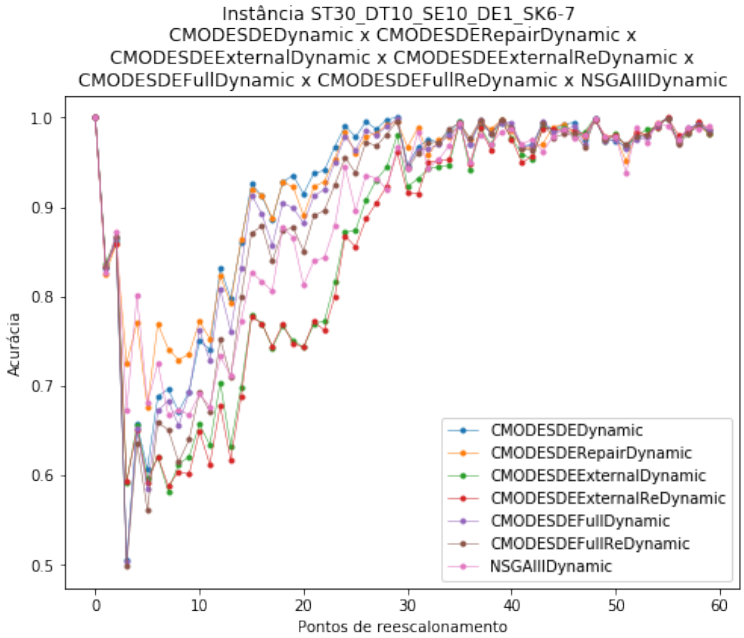

Figura 4.29 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I17

entre os algoritmos para a estabilidade, apresentando um valor-*p* de 2,583556E-01.

Na comparação pareada em relação ao hipervolume, Tabela 4.25, temos que existe diferença estatística entre vários algoritmos. Na Figura 4.31, podemos perceber claramente que o único algoritmo que não apresenta diferença estatística com os demais é o CMODESDEFullReDynamic, enquanto que a diferença estatística entre o CMODESDEDynamic e o CMODESDEExternalReDynamic apresenta um nível de confiança de mais de 99,9%.

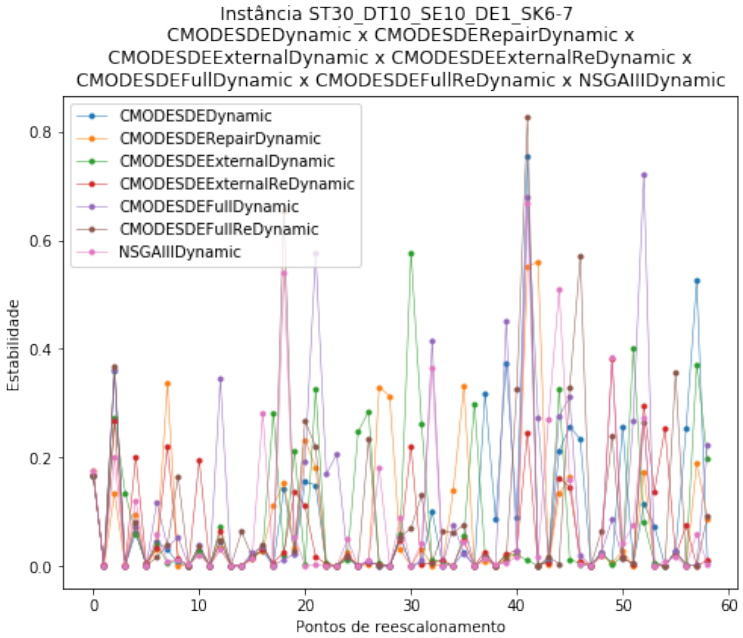

Figura 4.30 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I17

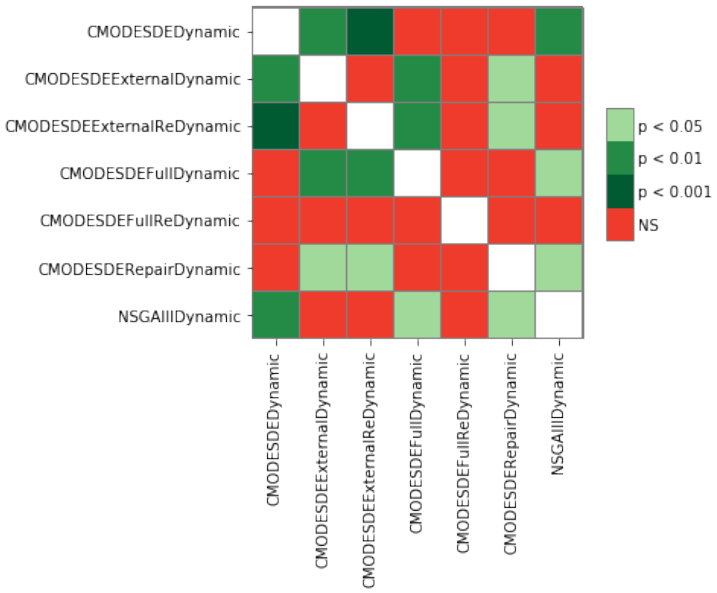

Figura 4.31 – Comparativo da significância indicada pelo valor-*p* entre todos os algoritmos dinâmicos para a instância I17

Tabela 4.25 – Comparação pareada dos valores-*p* obtidos na execução dos algoritmos dinâmicos para a instância I17

	CMODESDE-D	CMODESDE-ED	CMODESDE-ERD	CMODESDE-FD	CMODESDE-FRD	CMODESDE-RD	NSGAIII-D
CMODESDE-D	-1,00E+00	1,10E-03	8,89E-04	5,06E-01	1,45E-01	2,97E-01	1,36E-03
CMODESDE-ED	1,10E-03	-1,00E+00	9,50E-01	9,11E-03	6,85E-02	2,55E-02	9,51E-01
CMODESDE-ERD	8,89E-04	9,50E-01	-1,00E+00	7,60E-03	5,97E-02	2,17E-02	9,02E-01
CMODESDE-FD	5,06E-01	9,11E-03	7,60E-03	-1,00E+00	4,28E-01	7,06E-01	1,09E-02
CMODESDE-FRD	1,45E-01	6,85E-02	5,97E-02	4,28E-01	-1,00E+00	6,78E-01	7,83E-02
CMODESDE-RD	2,97E-01	2,55E-02	2,17E-02	7,06E-01	6,78E-01	-1,00E+00	2,97E-02
NSGAIII-D	1,36E-03	9,51E-01	9,02E-01	1,09E-02	7,83E-02	2,97E-02	-1,00E+00

De modo geral, os algoritmos com as técnicas dinâmicas apresentaram valores melhores que os algoritmos sem elas. As técnicas dinâmicas aplicadas ao CMODESDE foram melhores do que a aplicada ao NSGA-III. Os testes estatísticos pareados indicaram que existe diferença significativa entre os algoritmos em 11 das 18 instâncias. Nesse experimento, se repetiu o que tinha sido descoberto na QP3 onde ficou constatado que o CMODE tem um tempo de execução melhor que o NSGA-III. Além disso, a técnica que apresentou melhor desempenho em relação ao hipervolume e acurácia foi a de ajuste proativo, obtendo melhores resultados em 13 e 14, respectivamente, das 18 instâncias. Já para a estabilidade a técnica que apresentou melhores resultados foi a que utiliza um arquivo externo reavaliado. Curiosamente, a técnica de ajuste proativo não obteve melhor resultado em nenhuma instância quando verificada a estabilidade.

4.3 Considerações finais

A métrica mais utilizada para a comparação entre os algoritmos é o hipervolume, pois representa um melhor conjunto de soluções encontradas para o problema. Para um contexto dinâmico, temos a acurácia e a estabilidade, como medidas secundárias, umas vez que elas são baseadas no hipervolume.

Como mostrado nas seções anteriores, através dos experimentos realizados podemos constatar que a extensão do modelo proposto é viável aos algoritmos da literatura, principalmente quando comparados ao modelo de Shen *et al.* (2016). Além disso, três variações base do CMODE foram testadas para identificar qual obteve os melhores resultados, no qual não obtiveram diferenças estatísticas entre eles. Sendo assim, optamos por escolher o CMODESDE por ser um algoritmo com uma abordagem voltada para problemas com muitos objetivos e por ter alcançado valores melhores na métrica de hipervolume e igualado na acurácia.

Uma vez escolhido o algoritmo CMODESDE, foi realizado um teste para identificar se o CMODE é competitivo em relação a um algortimo muito utilizado na SBSE para problemas com muitos objetivos que é o NSGA-III. Nesse teste, percebemos que o CMODE é competitivo em relação ao NSGA-III, obtendo melhores resultados para todas as metricas na maioria das instâncias utilizadas, apesar de não apresentarem diferenças estatísticas. Com isso, podemos perceber que o DE se adequa muito bem a esse tipo

de problema, além de ser mais rápido que os algoritmos genéticos. Por fim, algumas abordagens dinâmicas foram utilizadas, como a detecção de mudança e abordagens de memória, com o intuito de melhorar os resultados do algoritmo. As técnicas que se sobressaíram foram as de ajuste proativo, através da detecção de mudança, para as métricas de hipervolume e acurácia, e a técnica de arquivo externo reavaliado, atráves da abordagem de memória, para a métrica de estabilidade.

5 Conclusão

Nesta dissertação, foi investigada a aplicação de algoritmos de otimização com muitos objetivos ao problema de escalonamento dinâmico de projetos de software. Em particular, o algoritmo de evolução diferencial foi comparado com o NSGA-III, que é um algoritmo comumente utilizado em outros domínios da área de SBSE. O algoritmo de evolução diferencial investigado foi o CMODE, que é um algoritmo com múltiplas populações e parâmetros auto adaptativos, adequado para problemas de otimização dinâmica, como no caso do DSPSP. Tanto o CMODE como o NSGA-III ainda não haviam sido aplicados ao DSPSP. Variantes do algoritmo base, obtidas pela incorporação de técnicas de otimização dinâmica também foram explorados para o DSPSP.

Neste trabalho, também propusemos um novo modelo para o DSPSP, na realidade uma extensão do modelo proposto por Shen *et al.* (2016), incorporando novas características de um projeto de software real a este modelo, como a chegada de novos empregados, a remoção de tarefas, a ocorrência de eventos dinâmicos paralelos e a adição de um novo objetivo que considera a experiência dos empregados para a realização das tarefas.

Para realizar os experimentos no novo modelo, foi também estendido o *framework* spsp-jmetal de Amaral (2018) para o modelo de Shen *et al.* (2016). Para a geração das instâncias, o gerador de instâncias também foi modificado para que os dados adicionais do novo modelo fossem considerados.

Para a validação adequada do algoritmo utilizado para o DSPSP, foram realizadas quatro baterias de experimentos visando responder as quatro questões de pesquisa estabelecidas. Em relação à extensão do modelo proposto, os experimentos mostraram que ele além de ser mais completo que o modelo de Shen *et al.* (2016), também é mais eficiente. Os experimentos também sugeriram que a meta-heurística CMODE pode ser promissora em relação a um algoritmo genético como o NSGA-III, quando aplicado ao problema de escalonamento dinâmico de projeto de software. A variação do CMODE utilizada para a comparação com o NSGA-III, o CMODESDE, apresentou melhor desempenho em 15 das 18 instâncias para o hipervolume e a acurácia, e em 12 das 18

instâncias para a estabilidade. Apesar disso, os testes estatísticos de comparação pareada mostraram que não existe diferença estatística significativa entre os algoritmos. Já para os experimentos utilizando técnicas dinâmicas nesses algoritmos, as variações das técnicas utilizadas no CMODE também apresentaram melhores resultados em relação às técnicas utilizadas no NSGA-III, onde a técnica de reparação das soluções apresentou melhores resultados em 13 das 18 instâncias para o hipervolume e em 14 das 18 instâncias para a acurácia. Considerando a estabilidade, a técnica que apresentou melhores resultados (9 das 18 instâncias) foi a da utilização de um arquivo externo com reavaliação das soluções. De modo geral, o CMODE com técnicas dinâmicas apresentou melhores resultados do que o NSGA-III com técnica dinâmica. Além disso, um fator importante descoberto durante os experimentos foi que o CMODE é muito mais rápido do que o NSGA-III, podendo o seu tempo de execução chegar a ser até seis vezes mais rápido.

Os resultados deste trabalho revelaram algumas oportunidades de trabalhos futuros. Entre eles estão: a incorporação de outras abordagens dinâmicas no CMODE; inclusão de outras métricas de desempenho para otimizações dinâmicas, tais como *General Distance* para problemas dinâmicos (VD) e *Hypervolume difference* (HVD), para o desempenho da convergência e da qualidade da solução encontrada; e análise de desempenho em relação aos valores dos objetivos; integrar o *framework* com ferramentas de gerenciamento de projetos, como por exemplo o MS Project, entre outras ferramentas; criação de uma ferramenta que permita ao gerente de projeto escolher os cronogramas encontrados. Em relação à extensão do modelo temos: investigação de outras características de projeto de software que possam ser agregadas ao modelo proposto, deixando-o ainda mais perto do ambiente de projetos reais, como por exemplo, mudanças na ordem de precedência das tarefas; mais características relacionadas às tarefas e aos empregados, como por exemplo, data de vencimento das tarefas e cursos de treinamento também devem ser considerados.

Referências

AFZAL, W.; TORKAR, R.; FELDT, R. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, Elsevier, v. 51, n. 6, p. 957–976, 2009.

ALBA, E.; CHICANO, F. Management of Software Projects with GAs. In: ELSEVIER SCIENCE INC. *Proceedings of the 6th Metaheuristics International Conference (MIC'05)*. Vienna, Austria, 2005. p. 13–18.

ALBA, E.; CHICANO, J. F. Software project management with GAs. *Information Sciences*, Elsevier, v. 177, n. 11, p. 2380–2401, 2007.

ALVAREZ-VALDES, R.; CRESPO, E.; TAMARIT, J. M.; VILLA, F. A scatter search algorithm for project scheduling under partially renewable resources. *Journal of Heuristics*, Springer, v. 12, n. 1-2, p. 95–113, 2006.

AMARAL, R. O. M. d. *Otimização com Muitos Objetivos por Múltiplos Enxames Aplicada ao Escalonamento Dinâmico de Projetos de Software*. Dissertação (Mestrado em Ciência da Computação) — Pós-Graduação em Ciência da Computação, Universidade Federal de Sergipe, São Cristóvão, 2018.

ANGIRA, R.; SANTOSH, A. Optimization of dynamic systems: A trigonometric differential evolution approach. *Computers & Chemical Engineering*, Elsevier, v. 31, n. 9, p. 1055–1063, 2007.

ANTONIOL, G.; PENTA, M. D.; HARMAN, M. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. In: IEEE. *10th International Symposium on Software Metrics, 2004. Proceedings.* Chicago, Illinois, USA, 2004. p. 172–183.

ANTONIOL, G.; PENTA, M. D.; HARMAN, M. Search-based techniques for optimizing software project resource allocation. In: SPRINGER. *Genetic and Evolutionary Computation Conference (GECCO'04)*. Berlin, Heidelberg, 2004. p. 1425–1426.

ANTONIOL, G.; PENTA, M. D.; HARMAN, M. Search-based techniques applied to optimization of project planning for a massive maintenance project. In: IEEE. *21st IEEE International Conference on Software Maintenance (ICSM'05)*. Budapest, Hungary, 2005. p. 240–249.

BABU, B.; JEHAN, M. M. L. Differential evolution for multi-objective optimization. In: IEEE. *2003 Congress on Evolutionary Computation (CEC)*. Canberra, ACT, Australia, 2003. v. 4, p. 2696–2703.

BOEHM, B. W.; MADACHY, R.; STEECE, B. *et al. Software cost estimation with Cocomo II*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

BREST, J.; ZAMUDA, A.; BOSKOVIC, B.; MAUCEC, M. S.; ZUMER, V. Dynamic optimization using self-adaptive differential evolution. In: IEEE. *2009 Congress on Evolutionary Computation (CEC)*. Trondheim, Norway, 2009. p. 415–422.

BRUCKER, P.; DREXL, A.; MÖHRING, R.; NEUMANN, K.; PESCH, E. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, Elsevier, v. 112, n. 1, p. 3–41, 1999.

CÁMARA, M.; ORTEGA, J.; TORO, F. J. Parallel processing for multi-objective optimization in dynamic environments. In: IEEE. *2007 IEEE International Parallel and Distributed Processing Symposium*. Rome, Italy, 2007. p. 1–8.

CHANG, C. K.; CHAO, C.; NGUYEN, T. T.; CHRISTENSEN, M. Software project management net: a new methodology on software management. In: IEEE. *Proceedings. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac'98)(Cat. No. 98CB 36241)*. Vienna, Austria, 1998. p. 534–539.

CHEN, W.-N.; ZHANG, J. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, IEEE, v. 39, n. 1, p. 1–17, 2013.

CHICANO, F.; LUNA, F.; NEBRO, A. J.; ALBA, E. Using multi-objective metaheuristics to solve the software project scheduling problem. In: ACM. *Genetic and Evolutionary Computation Conference (GECCO'11)*. New York, NY, USA, 2011. p. 1915–1922.

COELLO, C. A. C.; LAMONT, G. B.; VELDHUIZEN, D. A. V. *et al. Evolutionary algorithms for solving multi-objective problems*. New York: Springer, 2007. v. 5.

COLANZI, T. E.; VERGILIO, S. R.; ASSUNÇÃO, W. K. G.; POZO, A. Search based software engineering: Review and analysis of the field in Brazil. *Journal of Systems and Software*, Elsevier, v. 86, n. 4, p. 970–984, 2013.

CRUZ, C.; GONZÁLEZ, J. R.; PELTA, D. A. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, Springer, v. 15, n. 7, p. 1427–1448, 2011.

DAS, S.; ABRAHAM, A.; KONAR, A. Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives. In: *Advances of computational intelligence in industrial systems*. [S.l.]: Springer, 2008. p. 1–38.

DEB, K.; JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, v. 18, n. 4, p. 577–601, 2014.

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 6, n. 2, p. 182–197, 2002.

DERRAC, J.; GARCÍA, S.; MOLINA, D.; HERRERA, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, Elsevier, v. 1, n. 1, p. 3–18, 2011.

DUGGAN, J.; BYRNE, J.; LYONS, G. J. A task allocation optimizer for software construction. *IEEE Software*, IEEE, v. 21, n. 3, p. 76–82, 2004.

DURILLO, J. J.; NEBRO, A. J. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, Elsevier, v. 42, n. 10, p. 760–771, 2011.

EBERHART, R.; KENNEDY, J. Particle swarm optimization. In: CITESEER. *Proceedings of the IEEE International Conference on Neural Networks*. Perth, Australia, 1995. v. 4, p. 1942–1948.

EHRGOTT, M. *Multicriteria optimization*. Berlin, Heidelberg: Springer Science & Business Media, 2005. v. 491.

FAN, H.-Y.; LAMPINEN, J. A trigonometric mutation operation to differential evolution. *Journal of Global Optimization*, Springer, v. 27, n. 1, p. 105–129, 2003.

FERRUCCI, F.; HARMAN, M.; SARRO, F. Search-based software project management. In: *Software Project Management in a Changing World*. Berlin, Heidelberg: Springer, 2014. p. 373–399.

FONSECA, C. M.; PAQUETE, L.; LÓPEZ-IBÁNEZ, M. An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE. *2006 IEEE International Conference on Evolutionary Computation*. Vancouver, BC, Canada, 2006. p. 1157–1163.

FÜLÖP, J. Introduction to decision making methods. *Laboratory of operations research and decision systems, Computer and automation institute, Hungarian Academy of Sciences*, Citeseer, v. 1, 2005.

HARMAN, M.; JIA, Y.; ZHANG, Y. Achievements, open problems and challenges for search based software testing. In: IEEE. *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. Graz, Austria, 2015. p. 1–12.

HARMAN, M.; JONES, B. F. Search-based software engineering. *Information and software Technology*, Elsevier, v. 43, n. 14, p. 833–839, 2001.

HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, ACM, v. 45, n. 1, p. 11, 2012.

HARMAN, M.; MCMINN, P.; SOUZA, J. T. D.; YOO, S. Search based software engineering: Techniques, taxonomy, tutorial. In: *Empirical Software Engineering and Verification*. Berlin, Heidelberg: Springer, 2012. p. 1–59.

HELBIG, M.; ENGELBRECHT, A. P. Performance measures for dynamic multi-objective optimisation algorithms. *Information Sciences*, Elsevier, v. 250, p. 61–81, 2013.

HERIČKO, M.; ŽIVKOVIČ, A.; ROZMAN, I. An approach to optimizing software development team size. *Information Processing Letters*, Elsevier, v. 108, n. 3, p. 101–106, 2008.

ISHIBUCHI, H.; AKEDO, N.; OHYANAGI, H.; NOJIMA, Y. Behavior of EMO algorithms on many-objective optimization problems with correlated objectives. In: IEEE. *2011 IEEE Congress of Evolutionary Computation (CEC)*. New Orleans, LA, USA, 2011. p. 1465–1472.

ISHIBUCHI, H.; TSUKAMOTO, N.; NOJIMA, Y. Evolutionary many-objective optimization. In: IEEE. *2008 3rd International Workshop on Genetic and Evolving Systems*. Witten-Bommerholz, Germany, 2008. p. 47–52.

JAIMES, A. L.; QUINTERO, L. V. S.; COELLO, C. A. C. Ranking methods in many-objective evolutionary algorithms. In: *Nature-Inspired Algorithms for Optimisation*. Berlin, Heidelberg: Springer, 2009. p. 413–434.

KANG, D.; JUNG, J.; BAE, D.-H. Constraint-based human resource allocation in software projects. *Software: Practice and Experience*, Wiley Online Library, v. 41, n. 5, p. 551–577, 2011.

KESSENTINI, M.; MKAOUER, M. W.; CINNÉIDE, M. Ó.; HAYASHI, S.; DEB, K. A robust multi-objective approach to balance severity and importance of refactoring opportunities. *Empirical Software Engineering*, Springer, v. 22, n. 2, p. 894–927, 2017.

LI, C. *et al. Benchmark generator for CEC 2009 competition on dynamic optimization*. Trondheim, Norway, 2008.

LUKE, S. *Essentials of Metaheuristics*. second. San Francisco, California, USA: Lulu, 2013. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

LUNA, F.; GONZÁLEZ-ÁLVAREZ, D. L.; CHICANO, F.; VEGA-RODRÍGUEZ, M. A. On the scalability of multi-objective metaheuristics for the software scheduling problem. In: IEEE. *2011 11th International Conference on Intelligent Systems Design and Applications*. Cordoba, Spain, 2011. p. 1110–1115.

LUNA, F.; GONZÁLEZ-ÁLVAREZ, D. L.; CHICANO, F.; VEGA-RODRÍGUEZ, M. A. The software project scheduling problem: A scalability analysis of multi-objective metaheuristics. *Applied Soft Computing*, Elsevier, v. 15, p. 136–148, 2014.

MCMINN, P. Search-based software test data generation: a survey. *Software testing, Verification and reliability*, Wiley Online Library, v. 14, n. 2, p. 105–156, 2004.

MENDES, R.; MOHAIS, A. S. DynDE: a differential evolution for dynamic optimization problems. In: IEEE. *2005 Congress on Evolutionary Computation (CEC)*. Edinburgh, Scotland, UK, 2005. v. 3, p. 2808–2815.

NAKAYAMA, H.; KANESHIGE, K.; TAKEMOTO, S.; WATADA, Y. An application of a multi-objective programming technique to construction accuracy control of cable-stayed bridges. *European Journal of Operational Research*, Elsevier, v. 87, n. 3, p. 731–738, 1995.

NGUYEN, T. T.; YANG, S.; BRANKE, J. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, Elsevier, v. 6, p. 1–24, 2012.

PMI. *A Guide to the Project Management Body of Knowledge (PMBOK guide)*. 5ª. ed. Newtown Square, PA: Project Management Institute, 2013.

PRESSMAN, R. S.; MAXIM, B. *Software Engineering: A Practitioner's Approach, 8th Ed*. New York, NY, USA: McGraw-Hill Science/Engineering/Math, 2014. ISBN 978-0-07-802212-8.

PRICE, K.; STORN, R. Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous space. *Technical Report, International Computer Science Institute*, 1995.

PURSHOUSE, R. C.; JALBĂ, C.; FLEMING, P. J. Preference-driven co-evolutionary algorithms show promise for many-objective optimisation. In: SPRINGER. *International Conference on Evolutionary Multi-Criterion Optimization*. Berlin, Heidelberg, 2011. p. 136–150.

RÄIHÄ, O. A survey on search-based software design. *Computer Science Review*, Elsevier, v. 4, n. 4, p. 203–249, 2010.

REZENDE, A. V.; SILVA, L.; BRITTO, A.; AMARAL, R. Software project scheduling problem in the context of search-based software engineering: A systematic review. *Journal of Systems and Software*, Elsevier, 2019.

SAATY, T. L.; VARGAS, L. G. Comparison of eigenvalue, logarithmic least squares and least squares methods in estimating ratios. *Mathematical modelling*, Elsevier, v. 5, n. 5, p. 309–324, 1984.

SCHUTZE, O.; LARA, A.; COELLO, C. A. C. On the influence of the number of objectives on the hardness of a multiobjective optimization problem. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 15, n. 4, p. 444–455, 2011.

SHEN, X.; MINKU, L. L.; BAHSOON, R.; YAO, X. Dynamic software project scheduling through a proactive-rescheduling method. *IEEE Transactions on Software Engineering*, IEEE, v. 42, n. 7, p. 658–686, 2016.

SHEN, X.-N.; MINKU, L. L.; MARTURI, N.; GUO, Y.-N.; HAN, Y. A q-learning-based memetic algorithm for multi-objective dynamic software project scheduling. *Information Sciences*, Elsevier, v. 428, p. 1–29, 2018.

SHTUB, A.; BARD, J. F.; GLOBERSON, S. *et al. Project management: Processes, methodologies, and economics*. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.

SOMMERVILLE, I. *Software Engineering, 8th Edition*. Harlow, England: Addison-Wesley, 2007. (International computer science series).

The Standish Group International. The chaos report. The Standish Group International, 2015.

VESTERSTROM, J.; THOMSEN, R. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In: IEEE. *2004 Congress on Evolutionary Computation (CEC)*. Portland, OR, USA, 2004. v. 2, p. 1980–1987.

WANG, J.; ZHANG, W.; ZHANG, J. Cooperative differential evolution with multiple populations for multiobjective optimization. *IEEE Transactions on Cybernetics*, IEEE, v. 46, n. 12, p. 2848–2861, 2016.

WU, X.; CONSOLI, P.; MINKU, L.; OCHOA, G.; YAO, X. An evolutionary hyper-heuristic for the software project scheduling problem. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. Cham, 2016. p. 37–47.

XIAO, J.; OSTERWEIL, L. J.; WANG, Q.; LI, M. Dynamic resource scheduling in disruption-prone software development environments. In: SPRINGER. *International Conference on Fundamental Approaches to Software Engineering*. Berlin, Heidelberg, 2010. p. 107–122.

ZHANG, Q.; LI, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 11, n. 6, p. 712–731, 2007.

ZHANG, Y.; FINKELSTEIN, A.; HARMAN, M. Search based requirements optimisation: Existing work and challenges. In: SPRINGER. *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Berlin, Heidelberg, 2008. p. 88–94.

ZHOU, A. *et al.* Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, Elsevier, v. 1, n. 1, p. 32–49, 2011.

ZITZLER, E.; KÜNZLI, S. Indicator-based selection in multiobjective search. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. Berlin, Heidelberg, 2004. p. 832–842.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), v. 103, 2001.

Apêndices

APÊNDICE A – Gráficos referentes a QP3 para as instâncias I1 a I7, I9 a I14 e I16 a I18

Este apêndice contém todos os gráficos gerados de comparação entre os hipervolumes, acurácias e estabilidades para as instâncias referente à QP3, que não foram detalhadamente discutidas no corpo da dissertação. Neles, podemos perceber que quanto maior o número de variáveis do problema, melhor o CMODESDE se sobressai em relação ao NSGA-III.

A.1 Instância I1 (`sT10_dT10_sE5_dE1_SK4-5`)

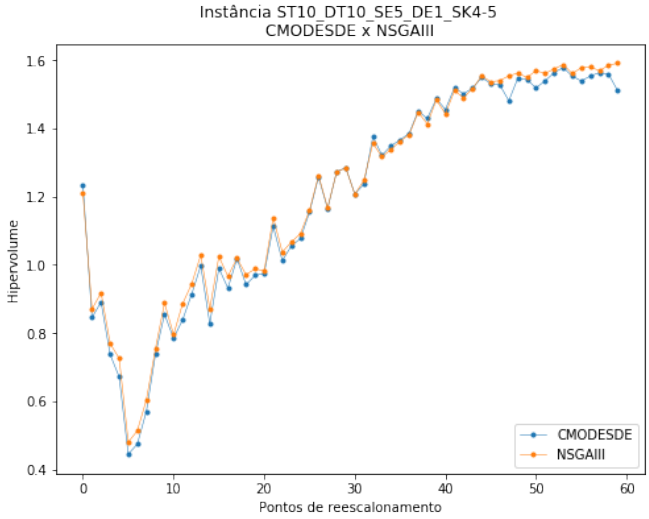

Figura A.1 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I1

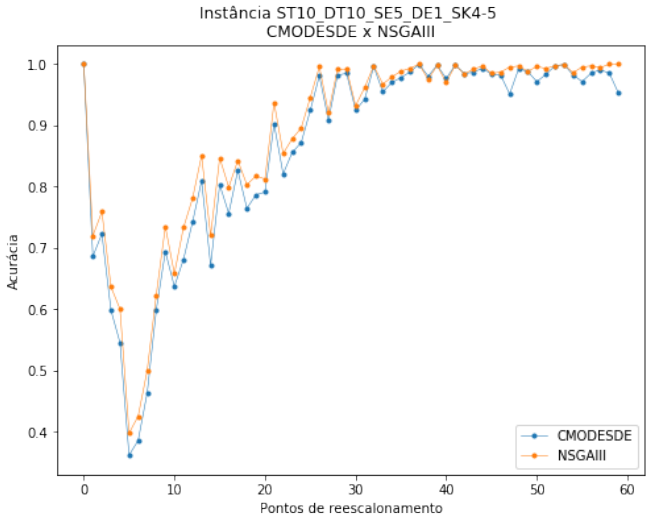

Figura A.2 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I1

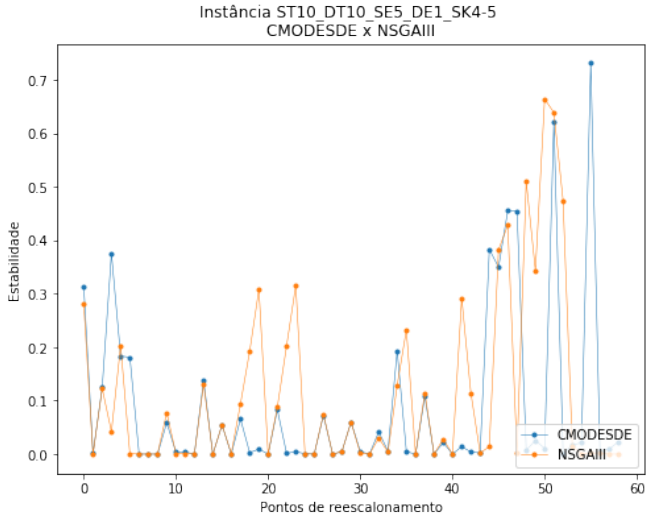

Figura A.3 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I1

A.2 Instância I2 (`sT10_dT10_sE10_dE1_SK4-5`)

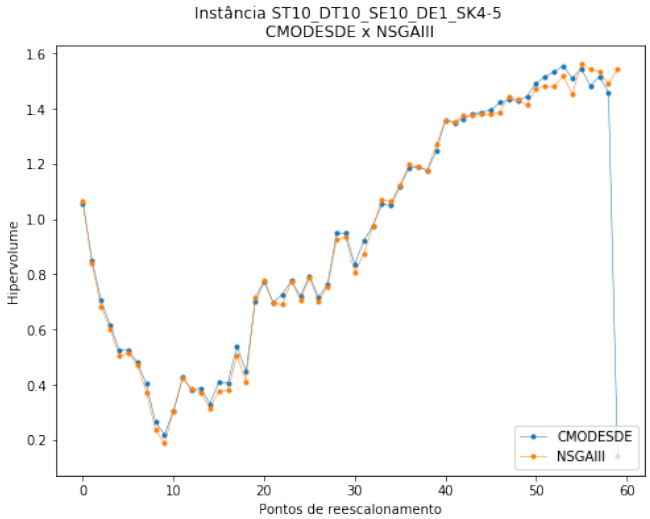

Figura A.4 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I2

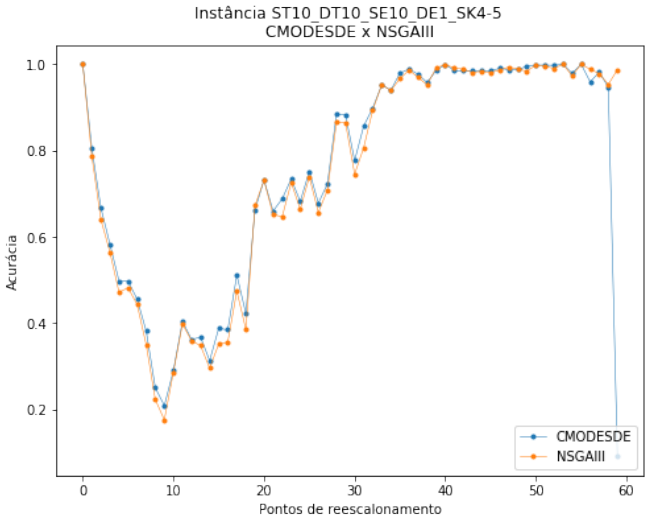

Figura A.5 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I2

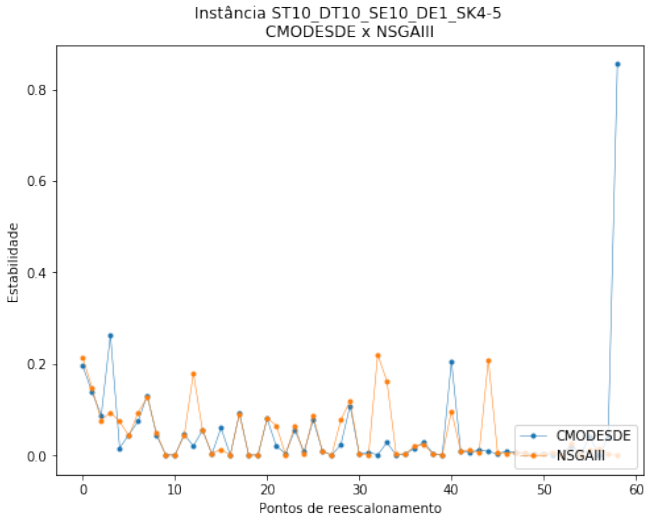

Figura A.6 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I2

A.3 Instância I3 (sT10_dT10_sE15_dE1_SK4-5)

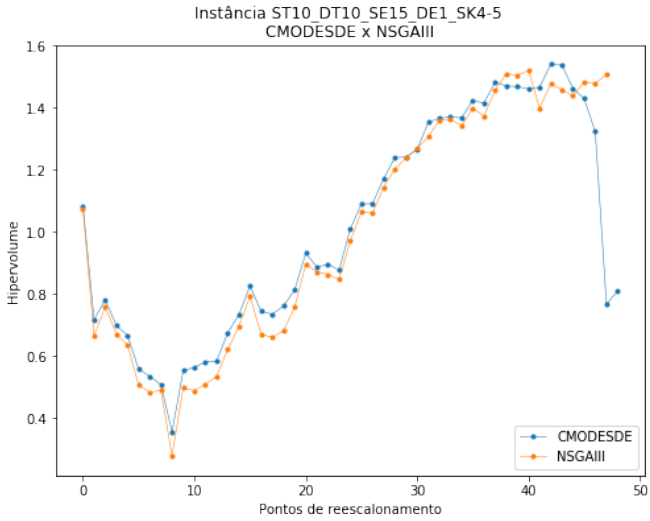

Figura A.7 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I3

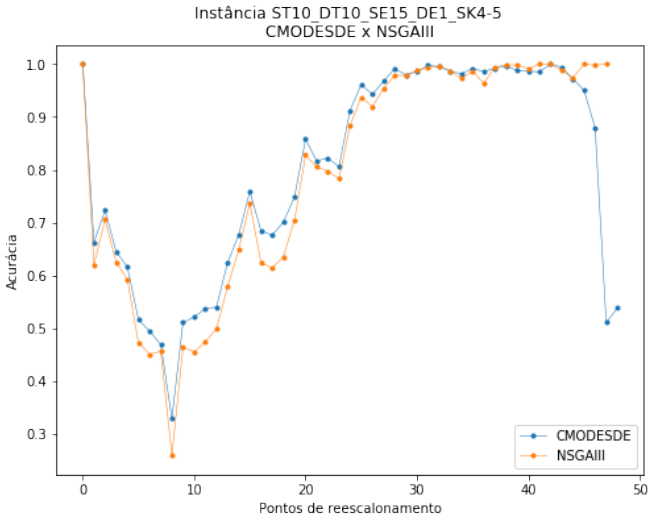

Figura A.8 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I3

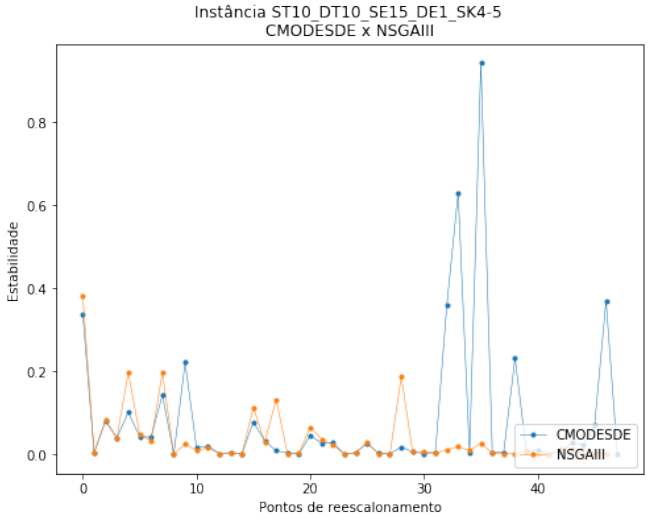

Figura A.9 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I3

A.4 Instância I4 (`sT10_dT10_sE5_dE1_SK6-7`)

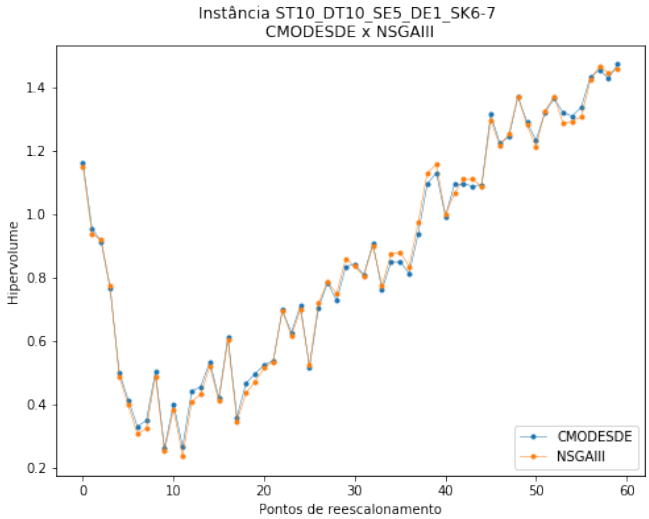

Figura A.10 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I4

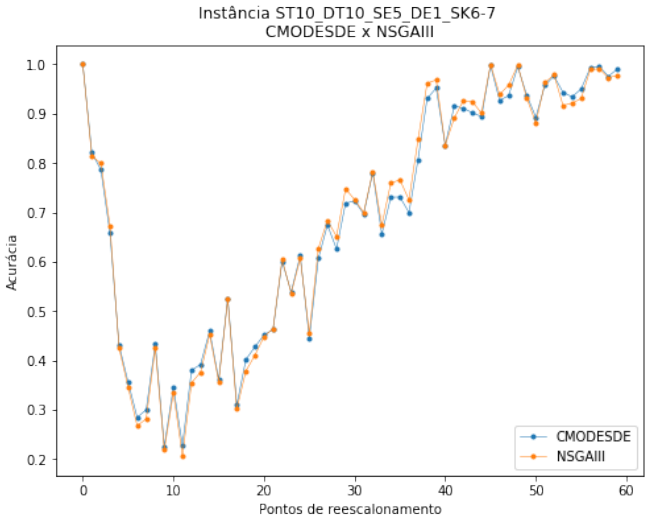

Figura A.11 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I4

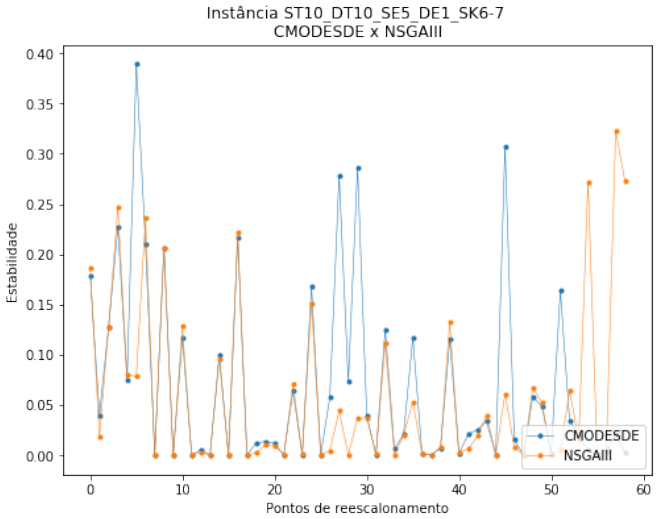

Figura A.12 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I4

A.5 Instância I5 (`sT10_dT10_sE10_dE1_SK6-7`)

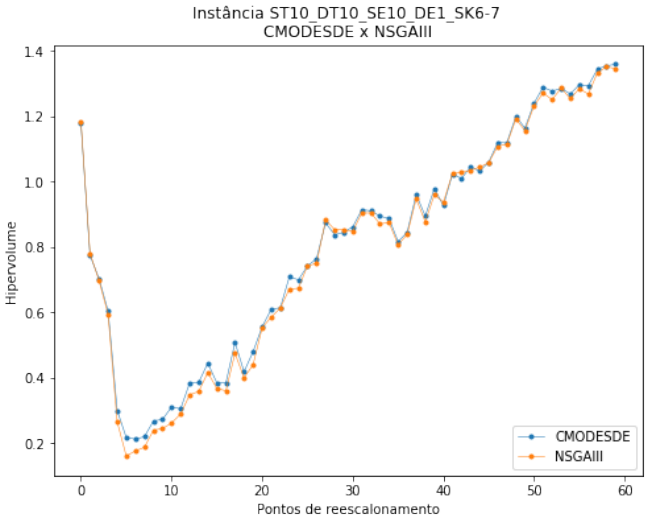

Figura A.13 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I5

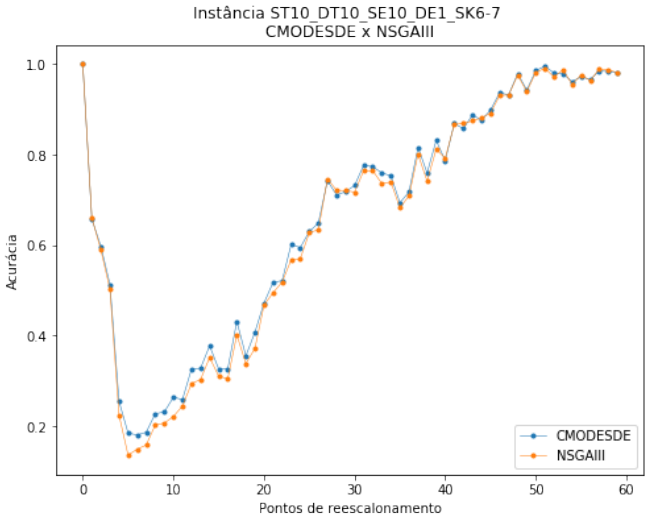

Figura A.14 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I5

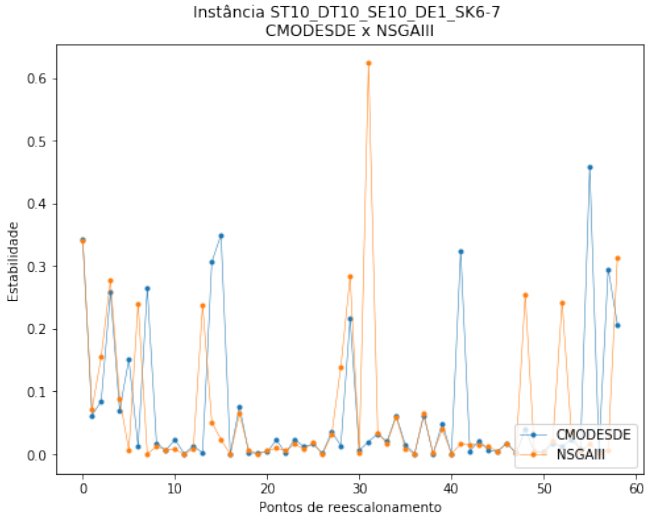

Figura A.15 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I5

A.6 Instância I6 (sT10_dT10_sE15_dE1_SK6-7)

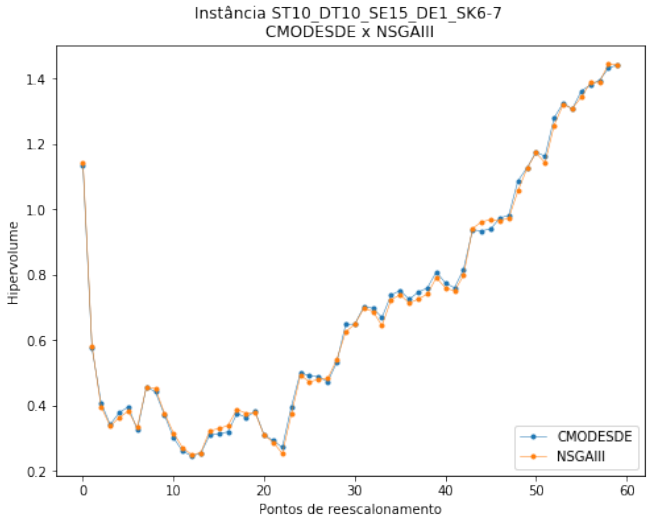

Figura A.16 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I6

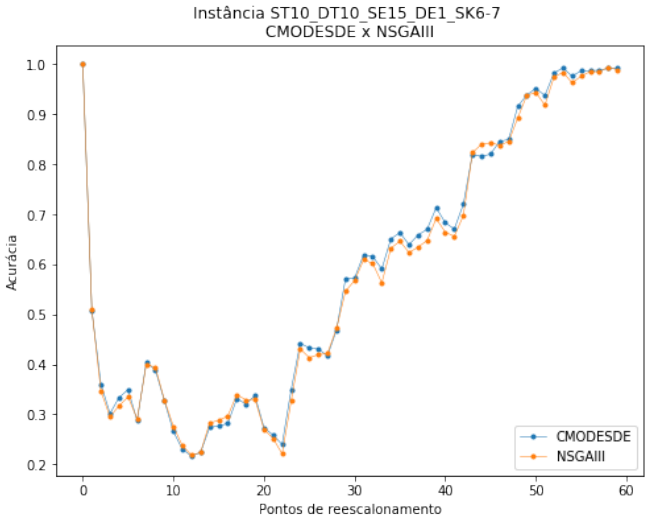

Figura A.17 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I6

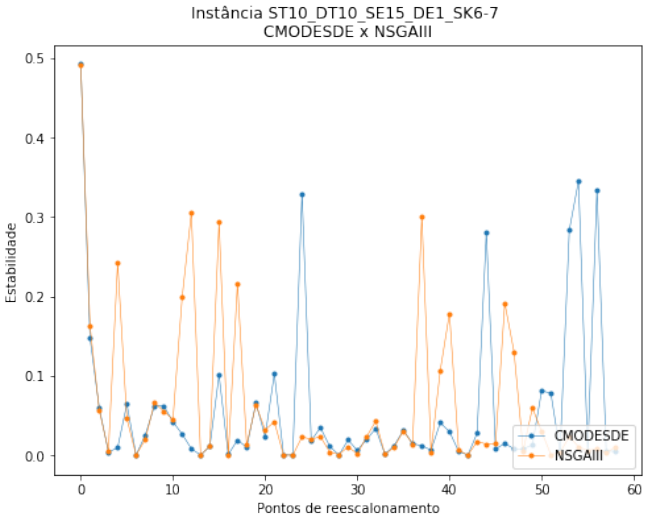

Figura A.18 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I6

A.7 Instância I7 (`sT20_dT10_sE5_dE1_SK4-5`)

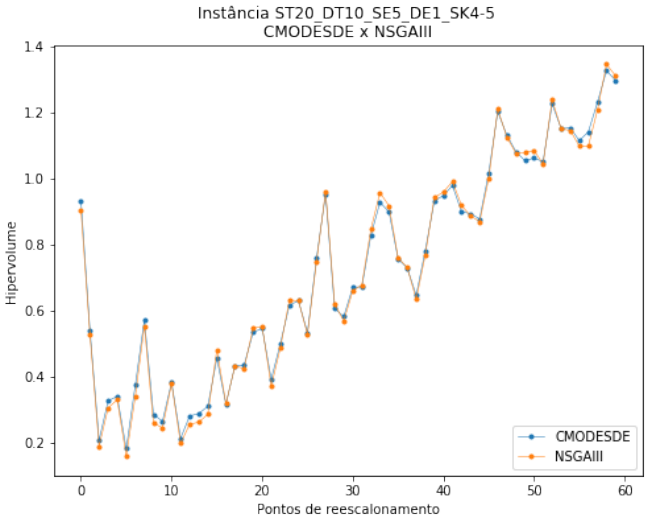

Figura A.19 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I7

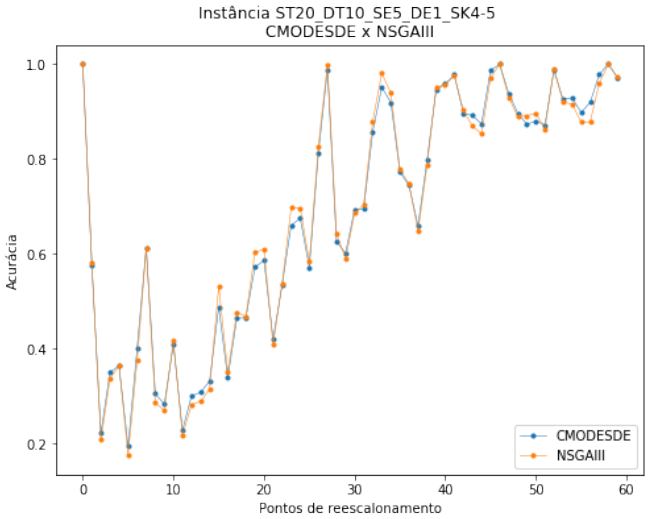

Figura A.20 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I7

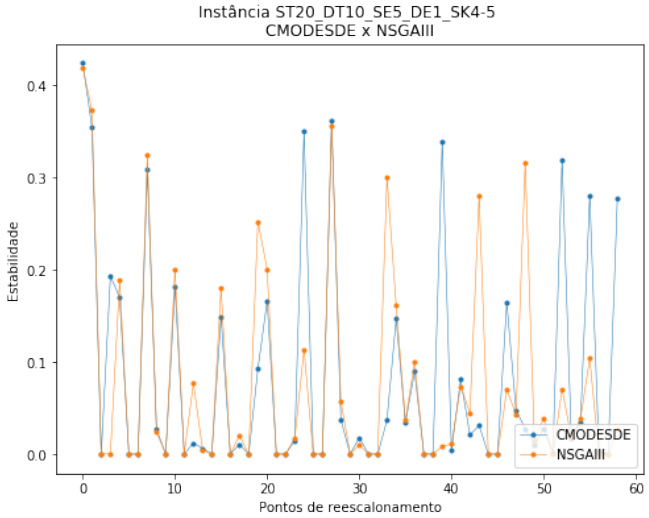

Figura A.21 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I7

A.8 Instância I9 (`sT20_dT10_sE15_dE1_SK4-5`)

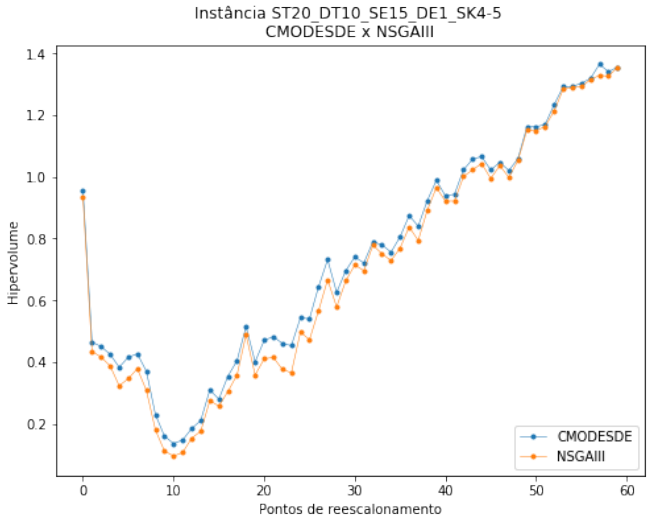

Figura A.22 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I9

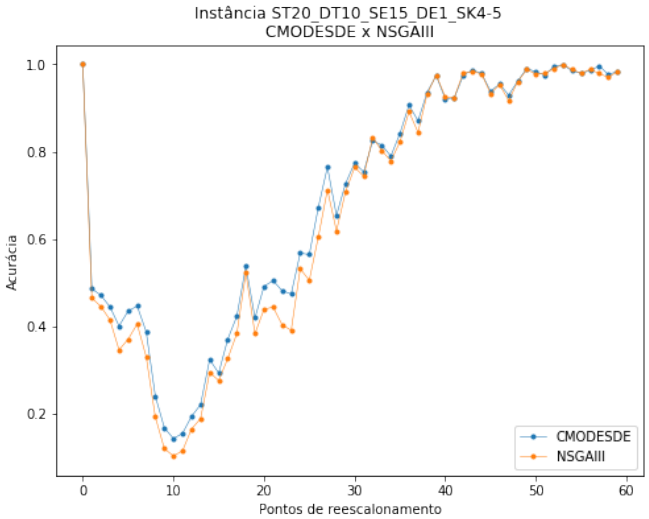

Figura A.23 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I9

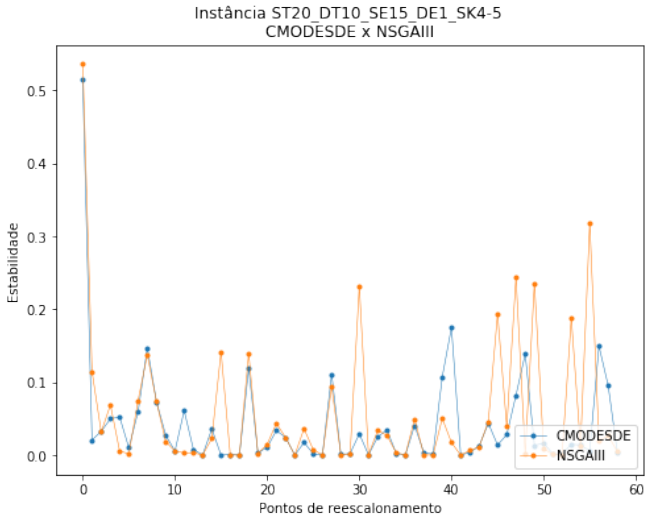

Figura A.24 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I9

A.9 Instância I10 (sT20_dT10_sE5_dE1_SK6-7)

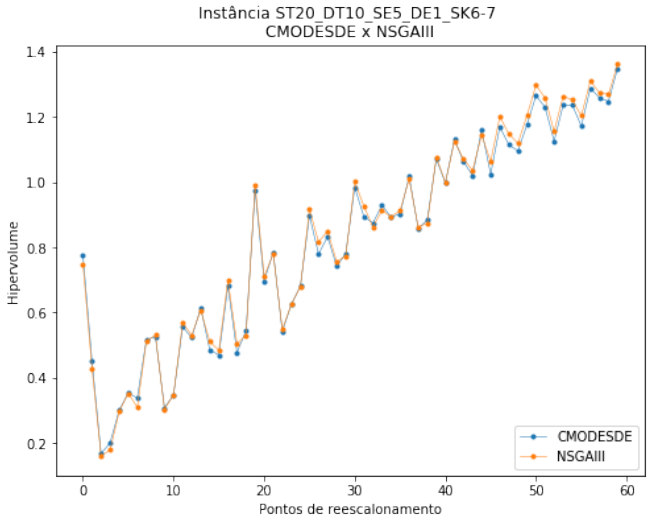

Figura A.25 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I10

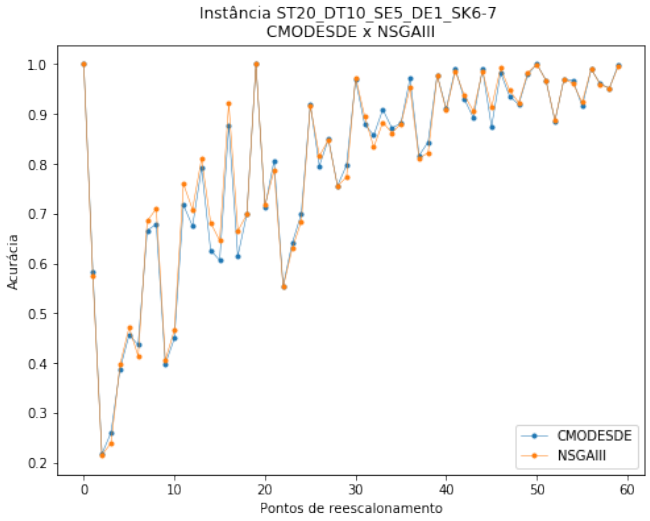

Figura A.26 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I10

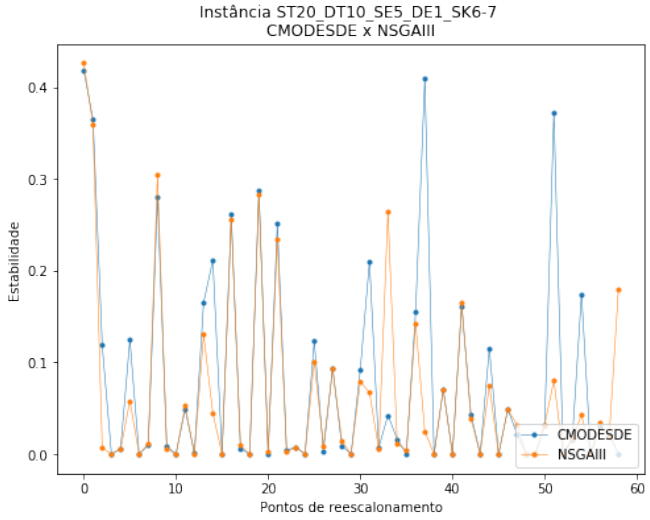

Figura A.27 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I10

A.10 Instância I11 (`sT20_dT10_sE10_dE1_SK6-7`)

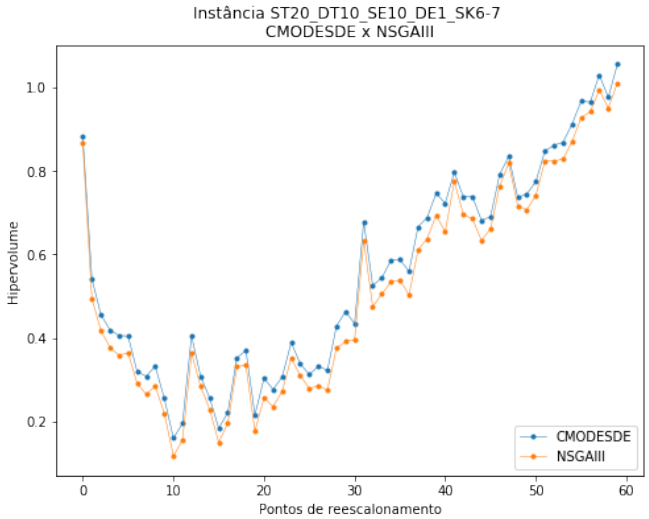

Figura A.28 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I11

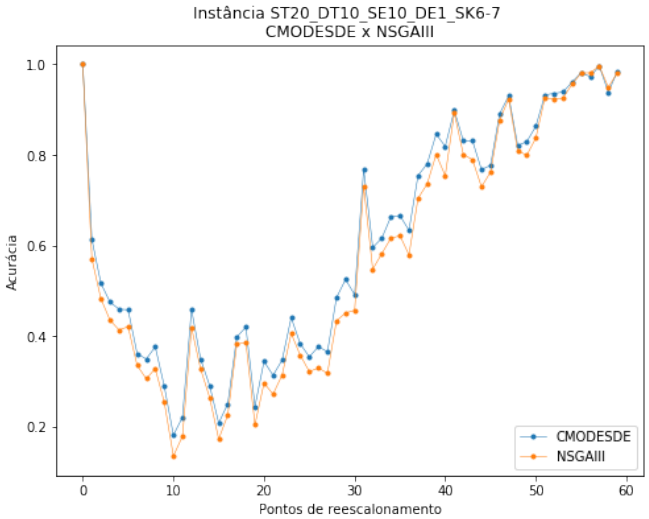

Figura A.29 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I11

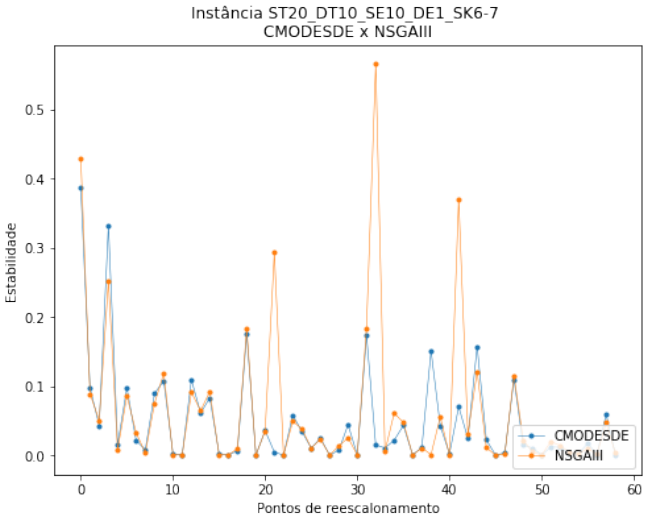

Figura A.30 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I11

A.11 Instância I12 (sT20_dT10_sE15_dE1_SK6-7)

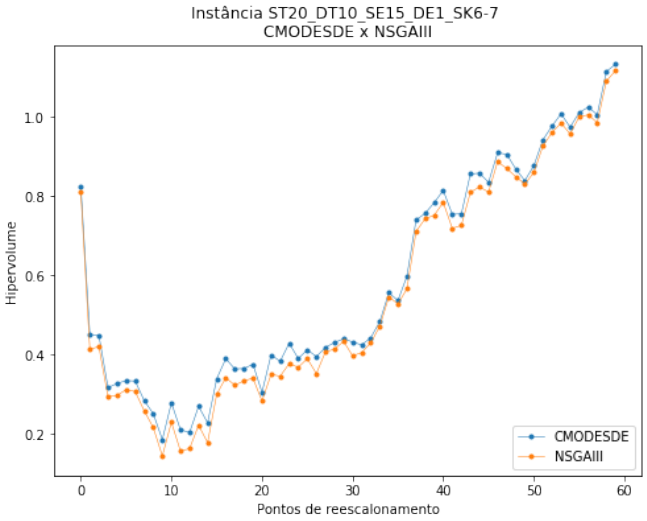

Figura A.31 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I12

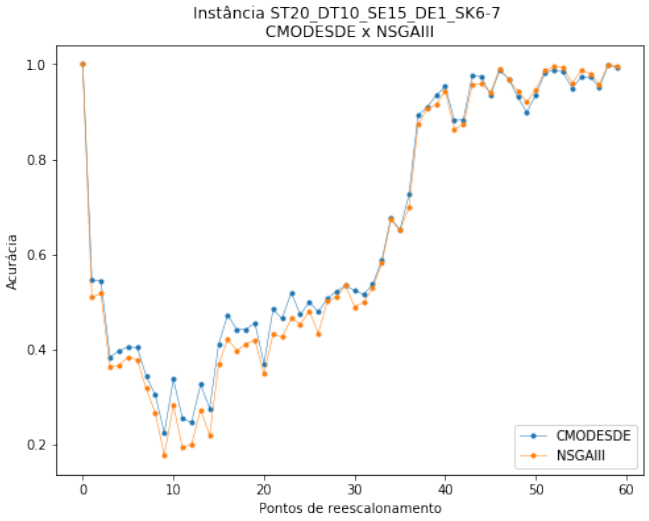

Figura A.32 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I12

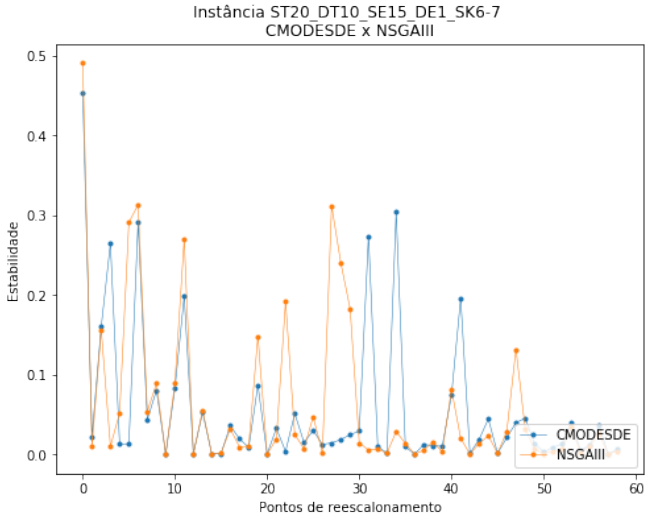

Figura A.33 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I12

A.12 Instância I13 (sT30_dT10_sE5_dE1_SK4-5)

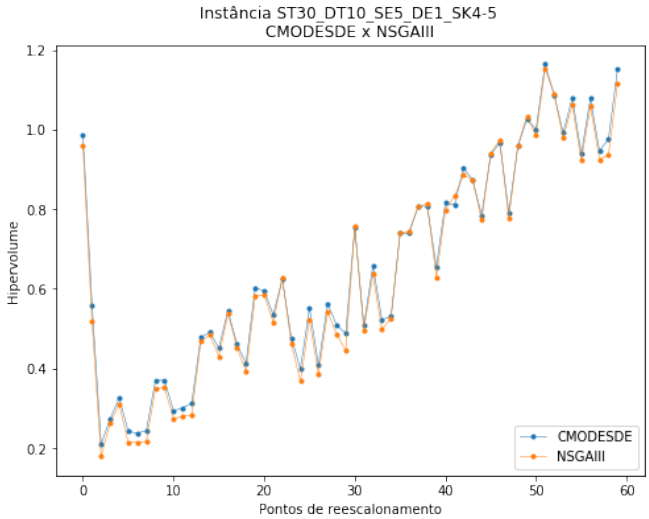

Figura A.34 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I13

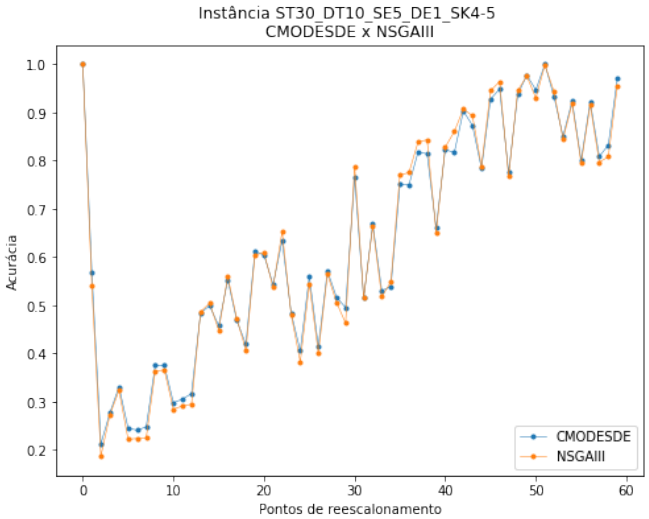

Figura A.35 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I13

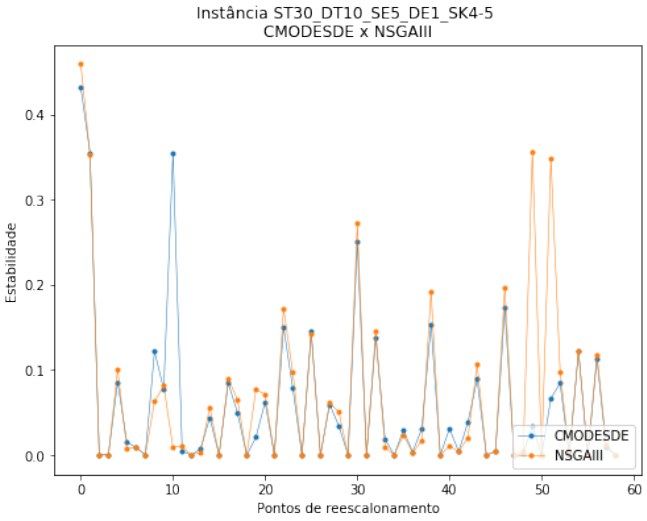

Figura A.36 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I13

A.13 Instância I14 (`sT30_dT10_sE10_dE1_SK4-5`)

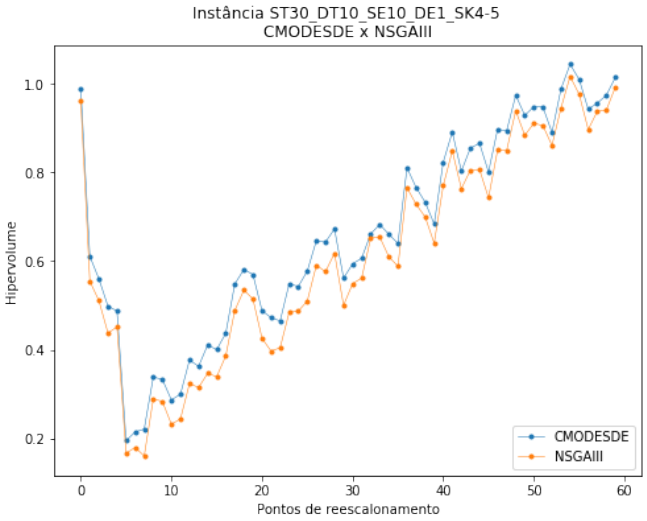

Figura A.37 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I14

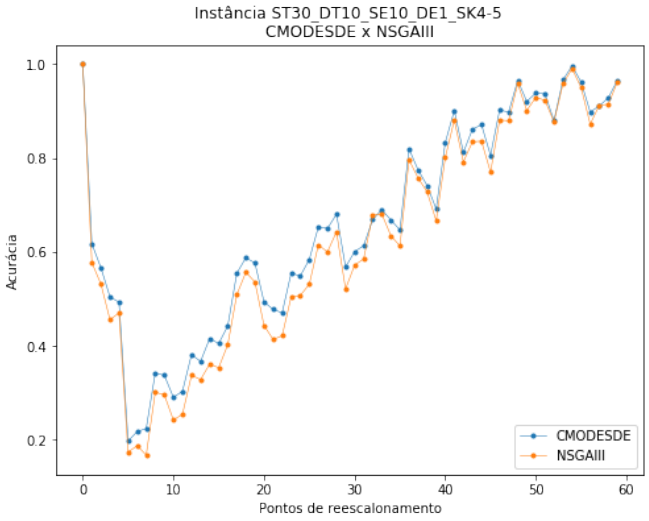

Figura A.38 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I14

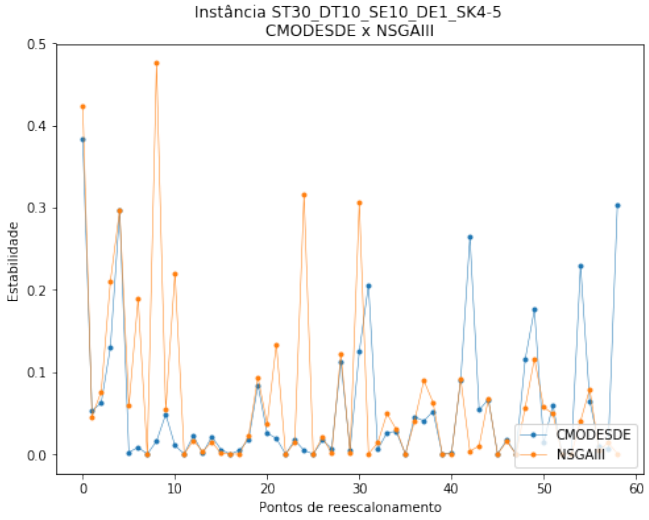

Figura A.39 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I14

A.14 Instância I16 (sT30_dT10_sE5_dE1_SK6-7)

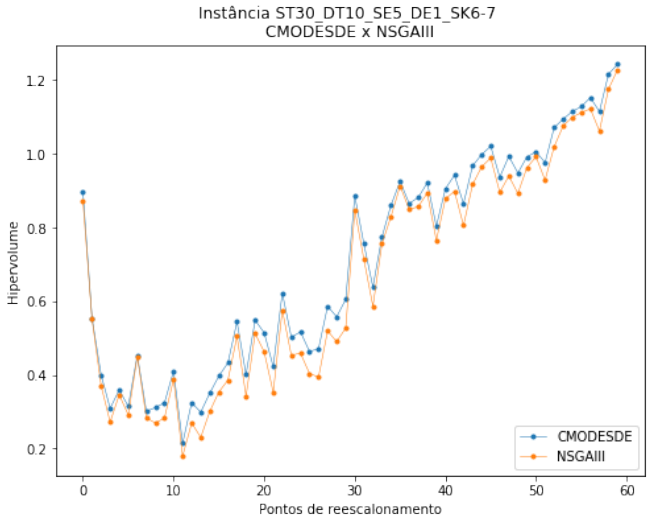

Figura A.40 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I16

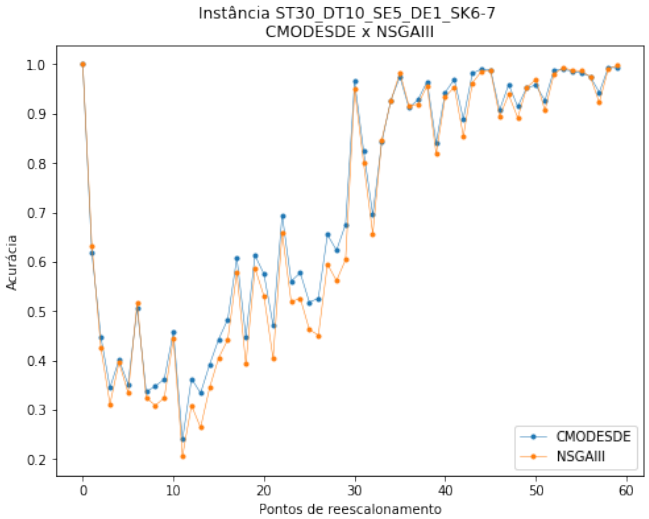

Figura A.41 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I16

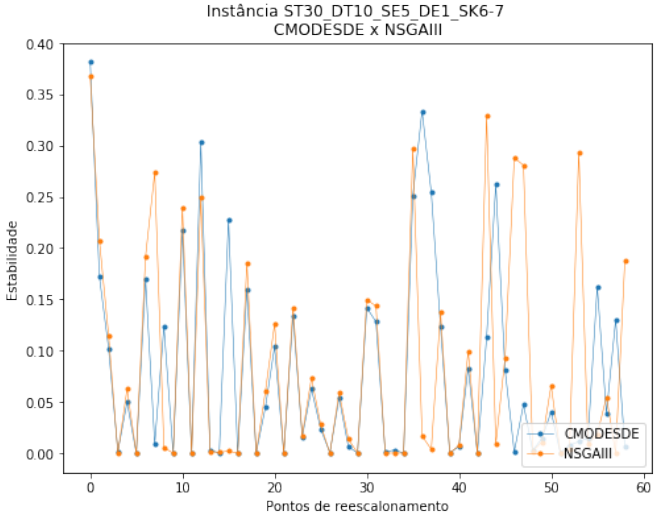

Figura A.42 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I16

A.15 Instância I17 (`sT30_dT10_sE10_dE1_SK6-7`)

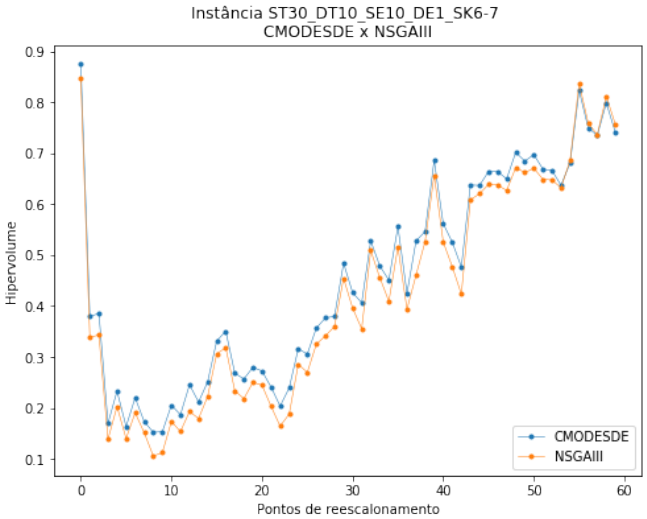

Figura A.43 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I17

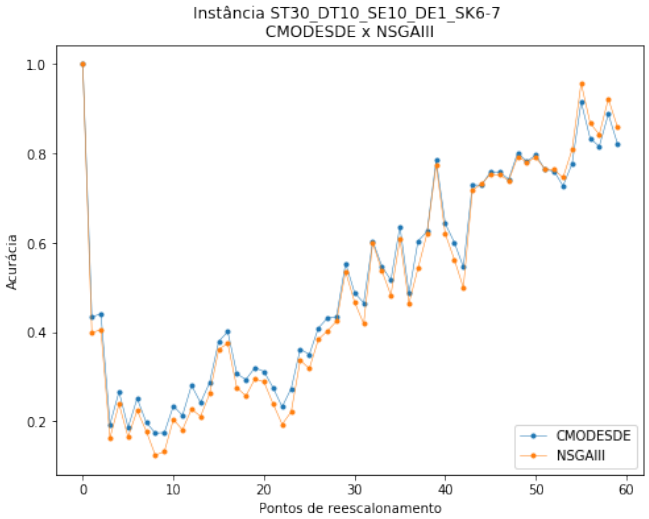

Figura A.44 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I17

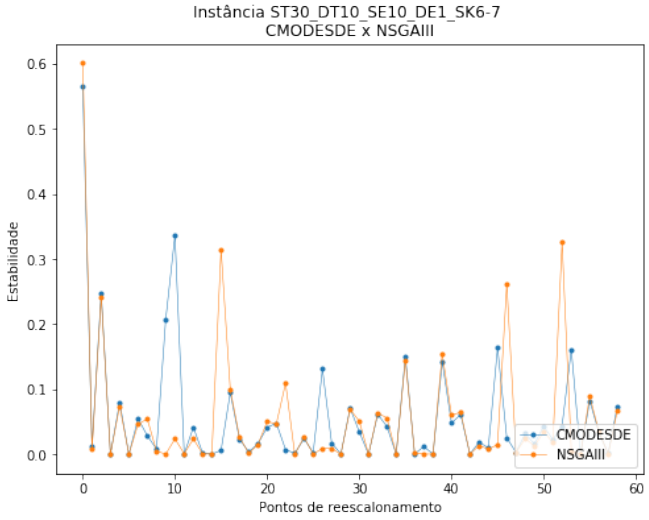

Figura A.45 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I17

A.16 Instância I18 (sT30_dT10_sE15_dE1_SK6-7)

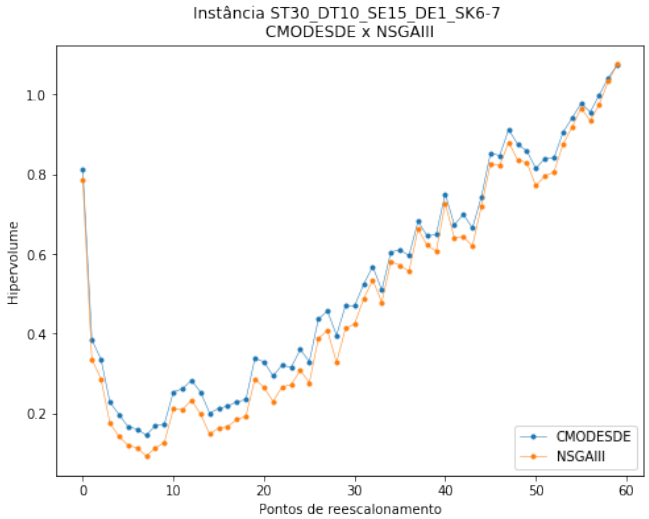

Figura A.46 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I18

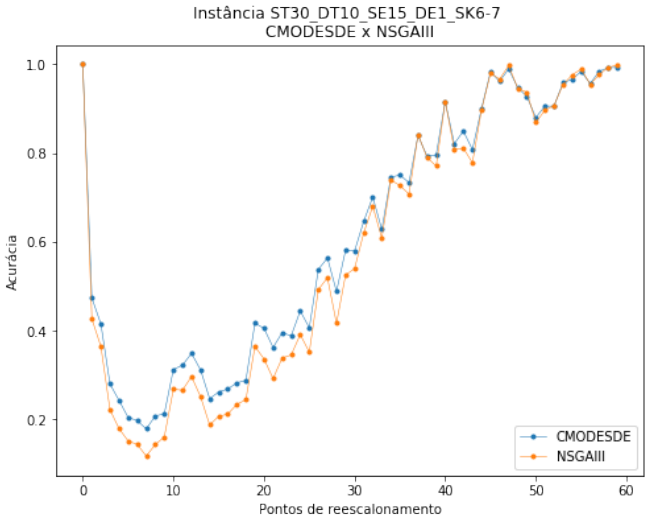

Figura A.47 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I18

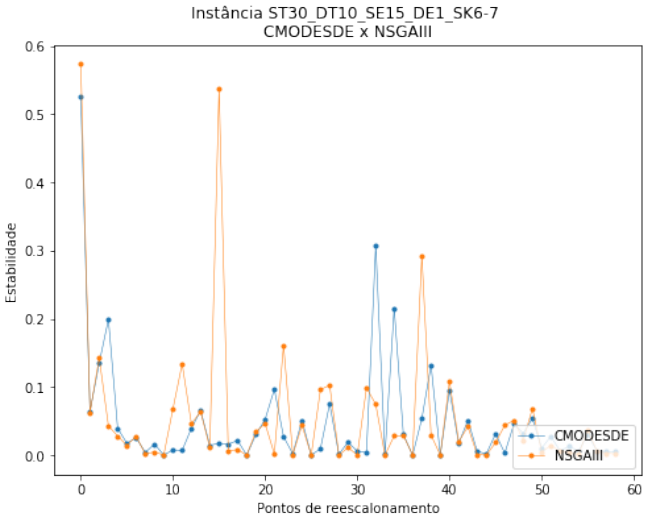

Figura A.48 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos CMODESDE e NSGA-III para a instância I18

APÊNDICE B – Gráficos referentes a QP4 para as instâncias I1 a I5, I8 a I16 e I18

Este apêndice contém todos os gráficos gerados de comparação entre os hipervolumes, acurácias e estabilidades para as instâncias referente à QP4, que não foram detalhadamente discutidas no corpo da dissertação. Neles, podemos perceber que o CMODESDERepairDynamic possui melhor desempenho que os demais algoritmos, na maioria das instâncias, nos 20 ou 30 primeiros pontos de reescalonamento, dependendo da instância.

B.1 Instância I1 (`sT10_dT10_sE5_dE1_SK4-5`)

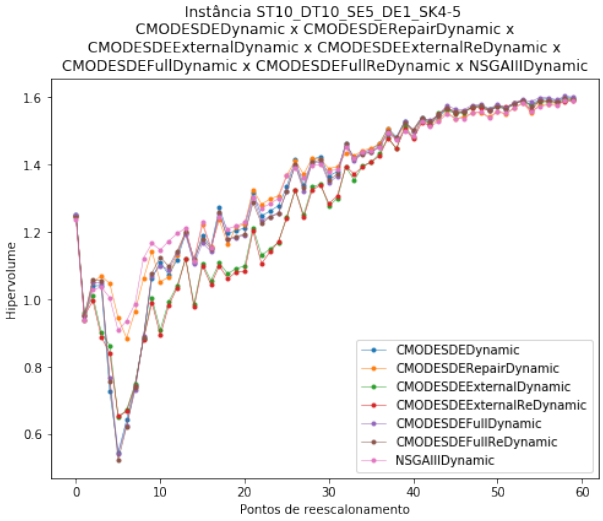

Figura B.1 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I1

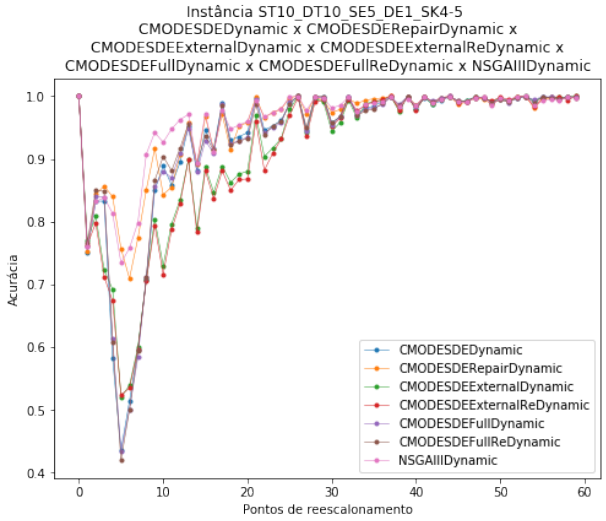

Figura B.2 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I1

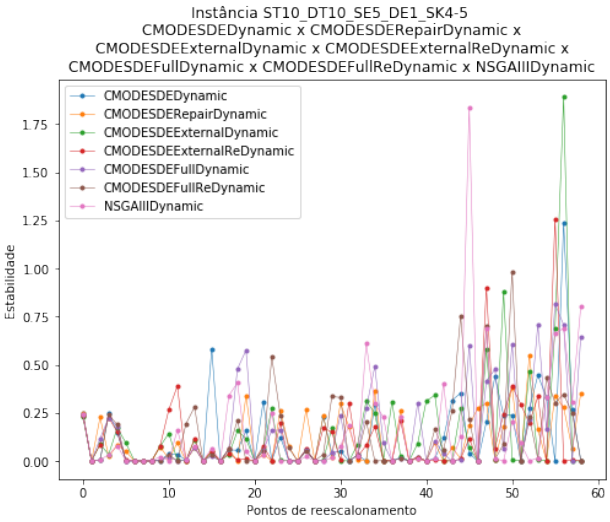

Figura B.3 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I1

B.2 Instância I2 (`sT10_dT10_sE10_dE1_SK4-5`)

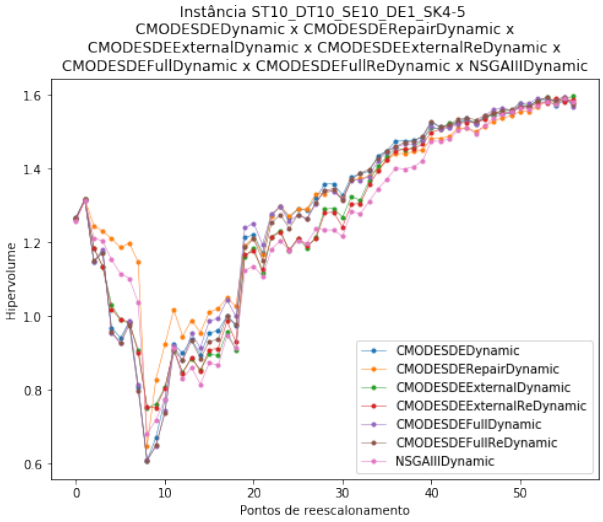

Figura B.4 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I2

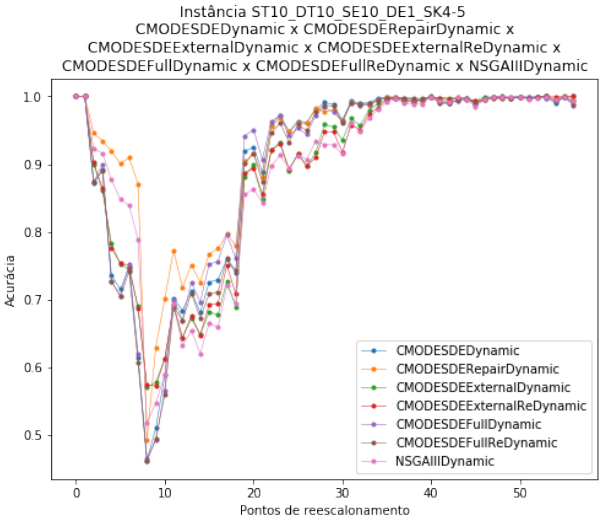

Figura B.5 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I2

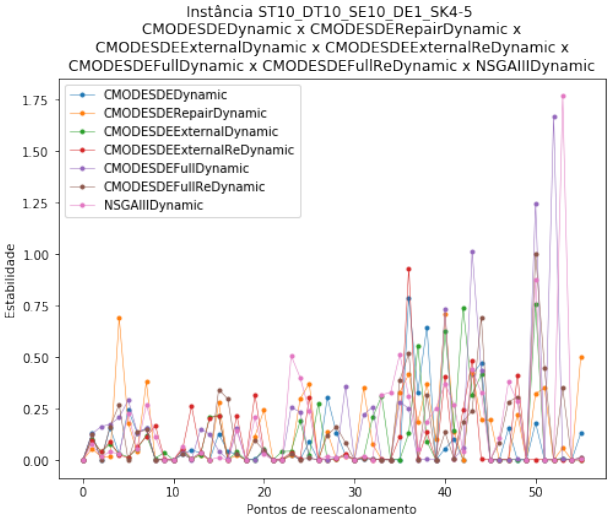

Figura B.6 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I2

B.3 Instância I3 (sT10_dT10_sE15_dE1_SK4-5)

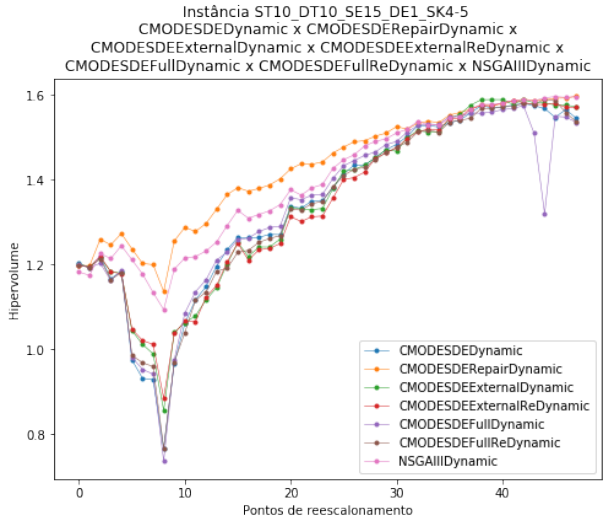

Figura B.7 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I3

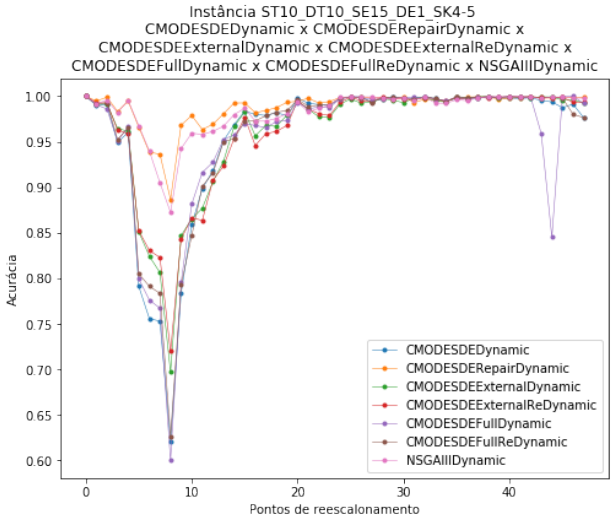

Figura B.8 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I3

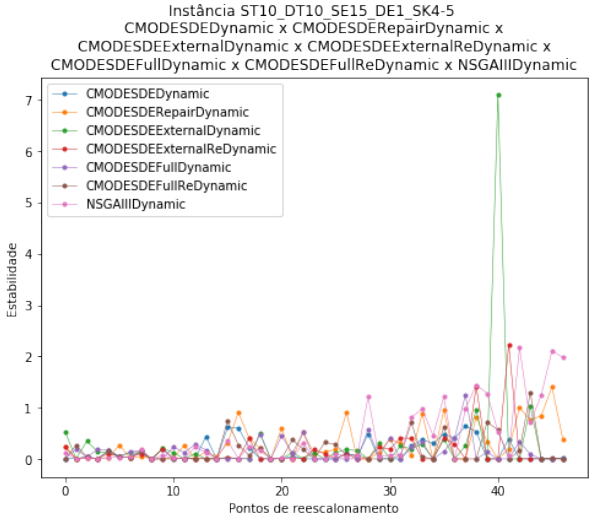

Figura B.9 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I3

B.4 Instância I4 (`sT10_dT10_sE5_dE1_SK6-7`)

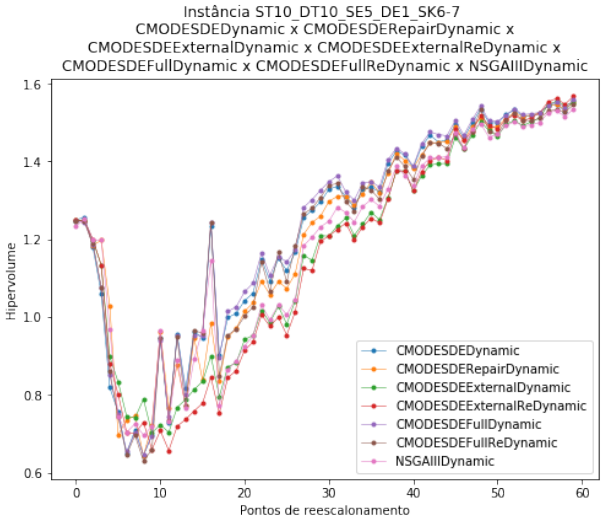

Figura B.10 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I4

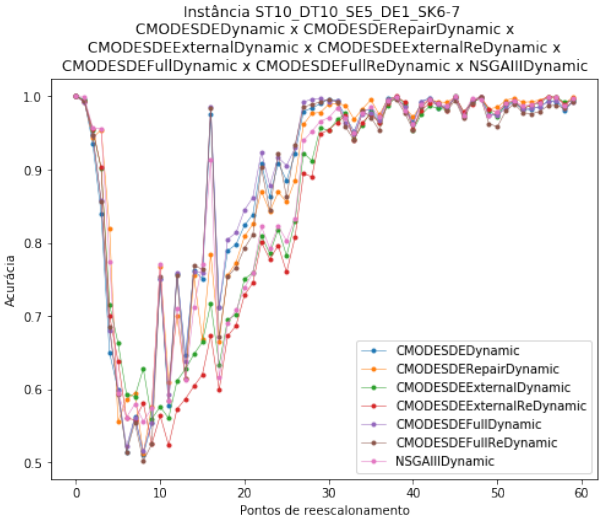

Figura B.11 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I4

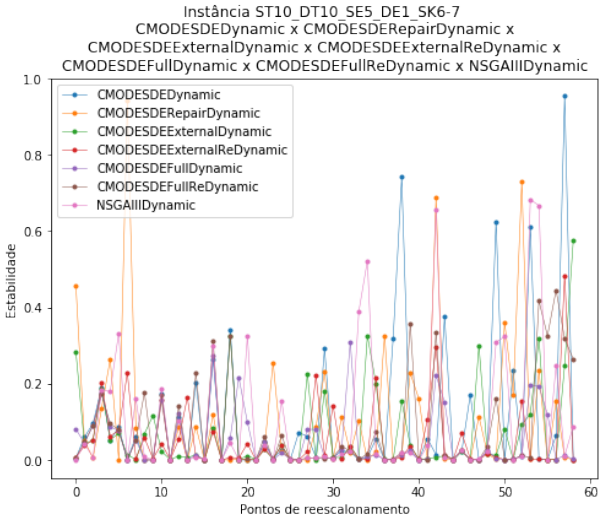

Figura B.12 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I4

B.5 Instância I5 (sT10_dT10_sE10_dE1_SK6-7)

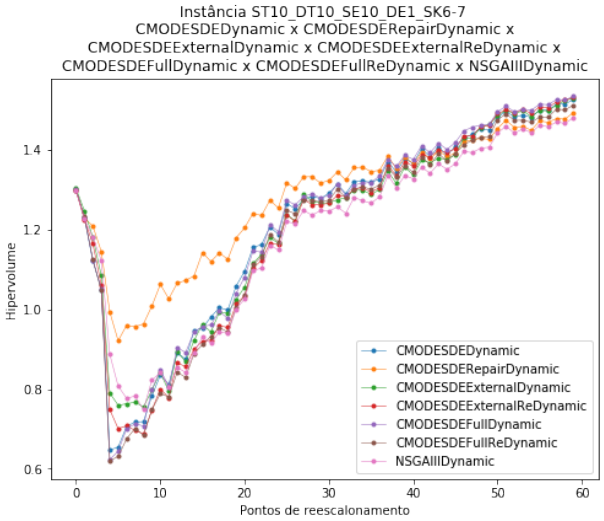

Figura B.13 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I5

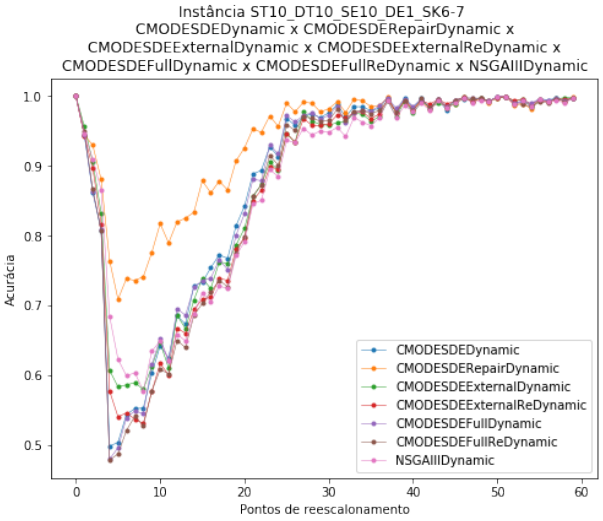

Figura B.14 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I5

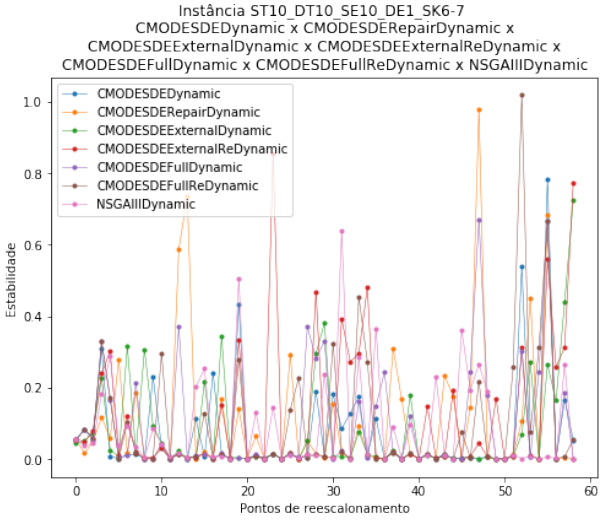

Figura B.15 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I5

B.6 Instância I8 (`sT20_dT10_sE10_dE1_SK4-5`)

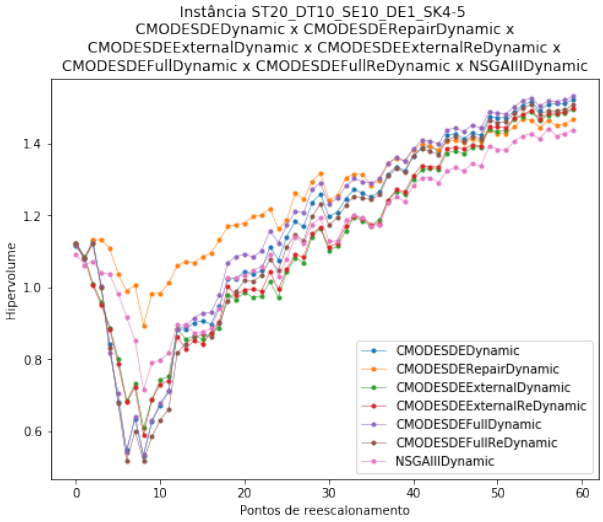

Figura B.16 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I8

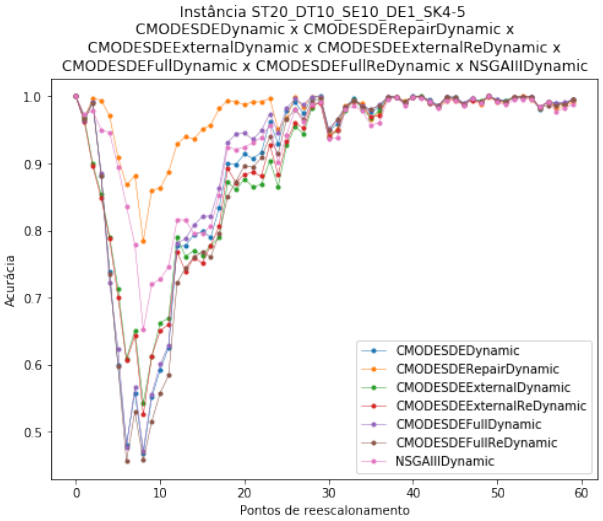

Figura B.17 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I8

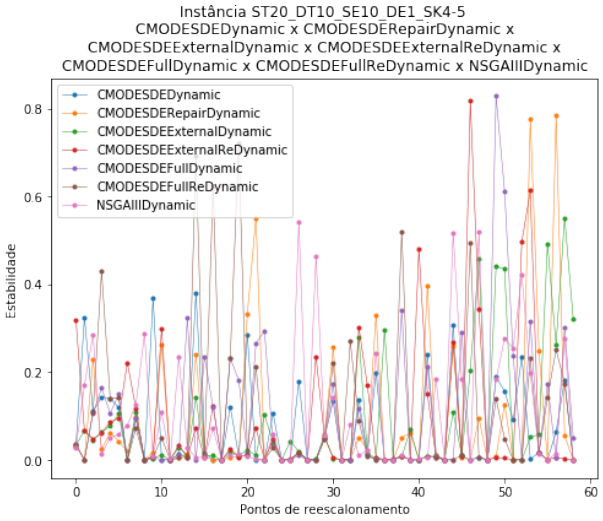

Figura B.18 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I8

B.7 Instância I9 (sT20_dT10_sE15_dE1_SK4-5)

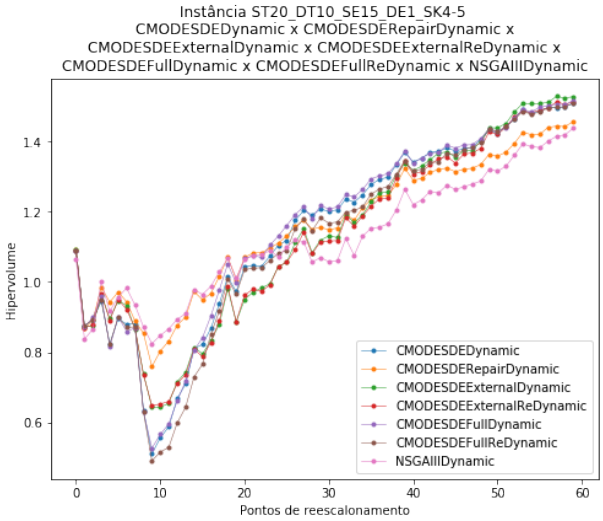

Figura B.19 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I9

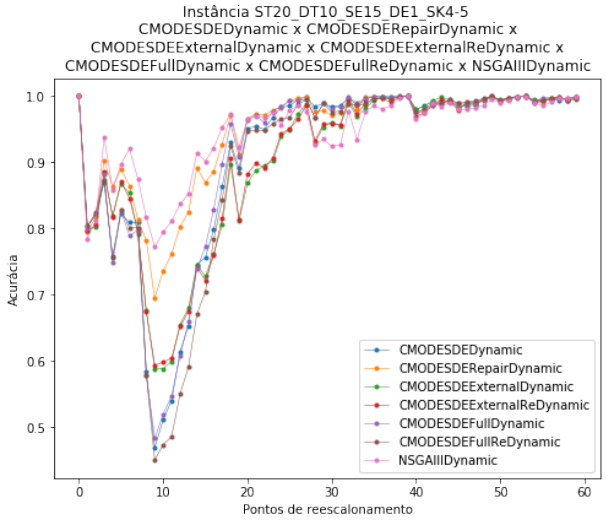

Figura B.20 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I9

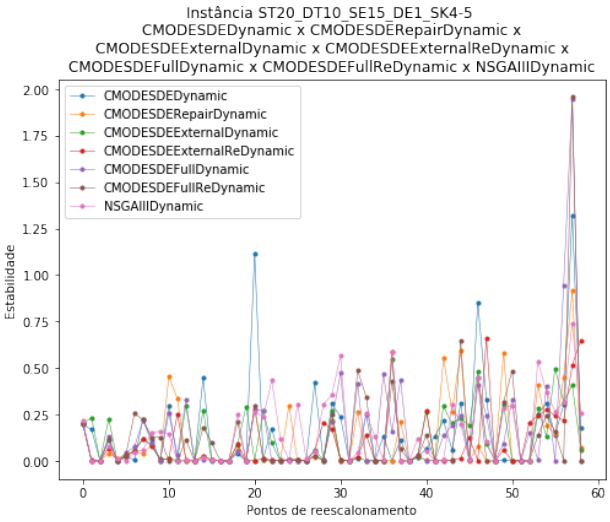

Figura B.21 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I9

B.8 Instância I10 (`sT20_dT10_sE5_dE1_SK6-7`)

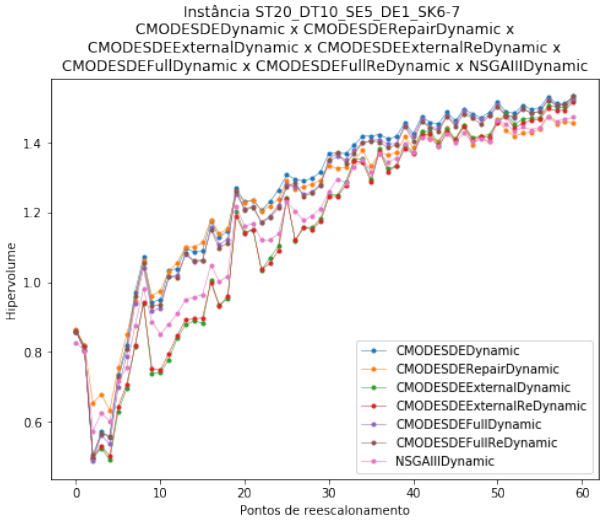

Figura B.22 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I10

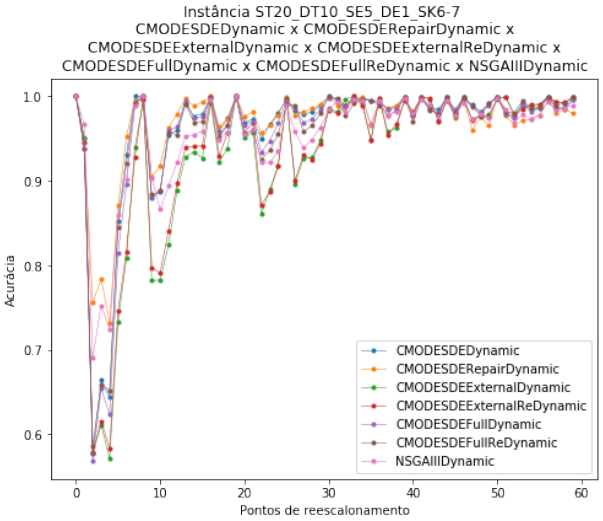

Figura B.23 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I10

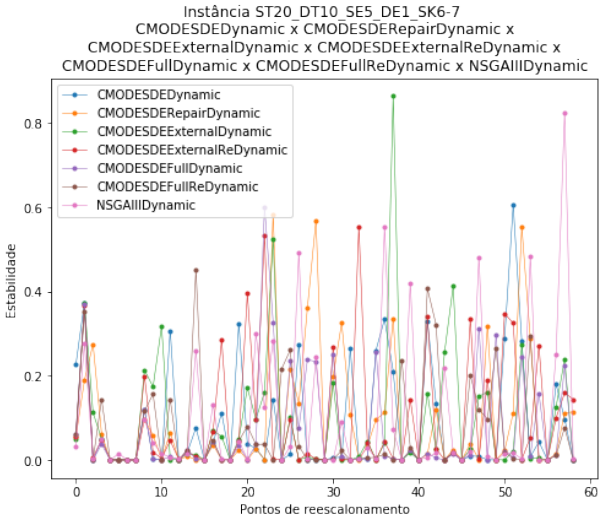

Figura B.24 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I10

B.9 Instância I11 (sT20_dT10_sE10_dE1_SK6-7)

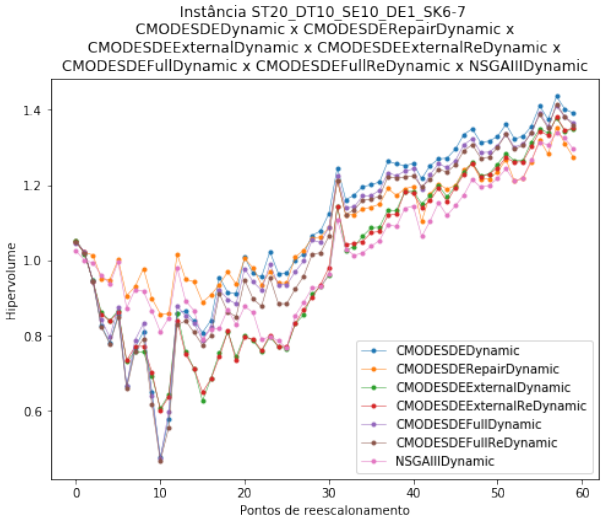

Figura B.25 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I11

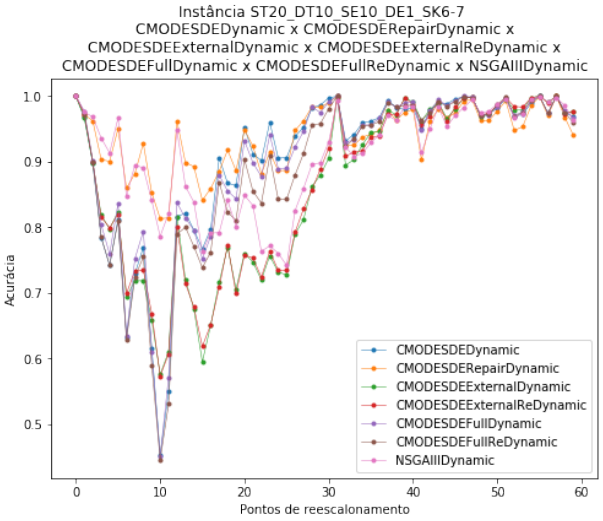

Figura B.26 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I11

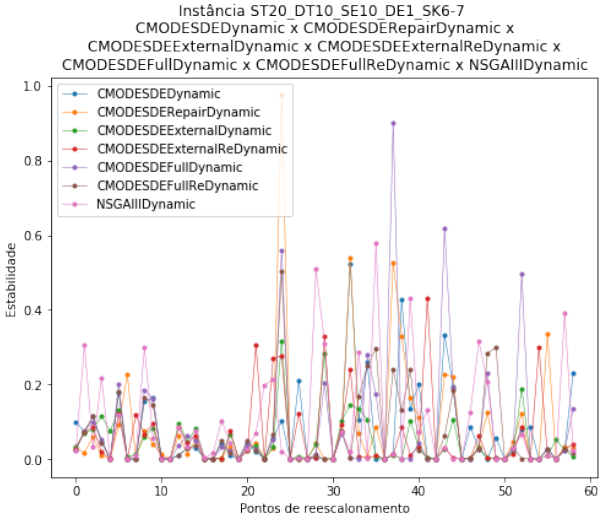

Figura B.27 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I11

B.10 Instância I12 (sT20_dT10_sE15_dE1_SK6-7)

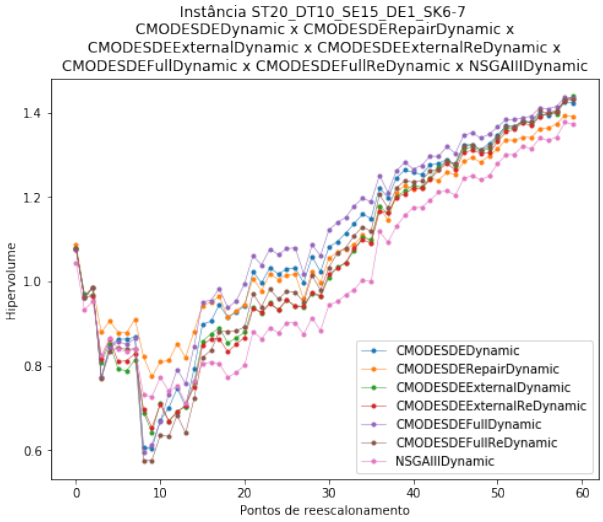

Figura B.28 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I12

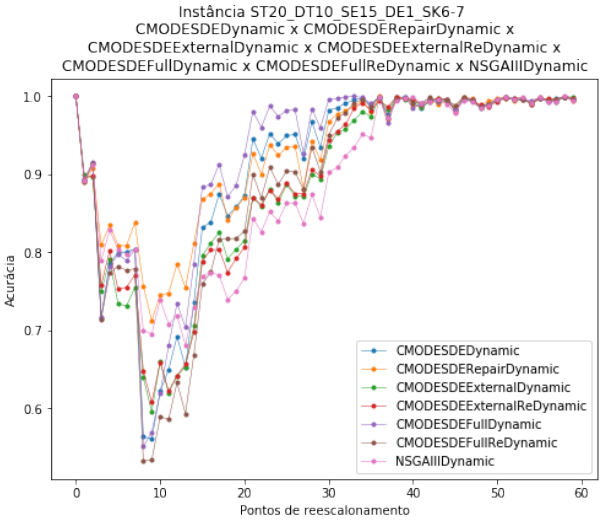

Figura B.29 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I12

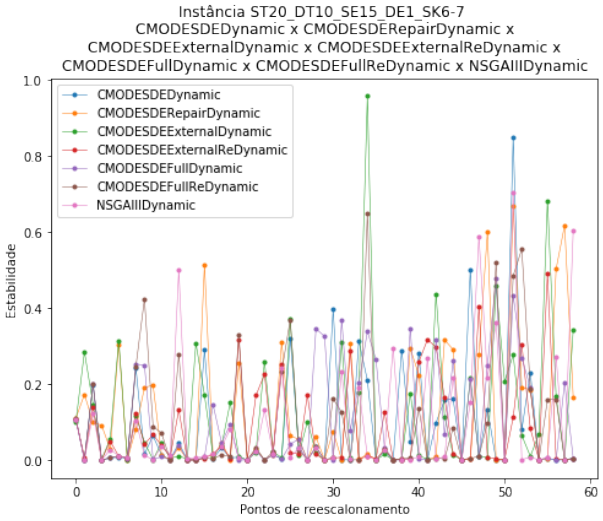

Figura B.30 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I12

B.11 Instância I13 (sT30_dT10_sE5_dE1_SK4-5)

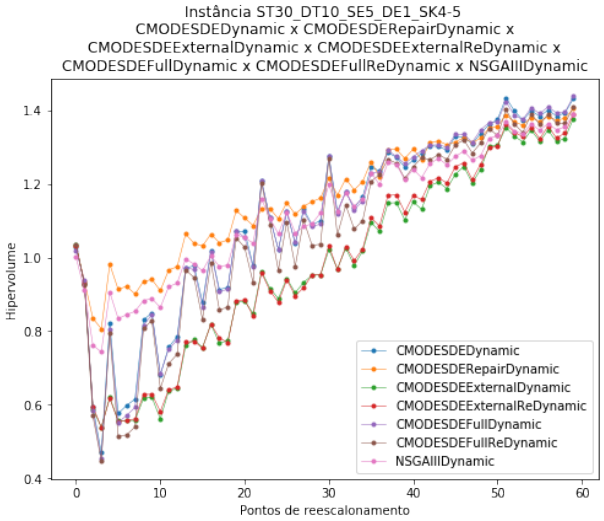

Figura B.31 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I13

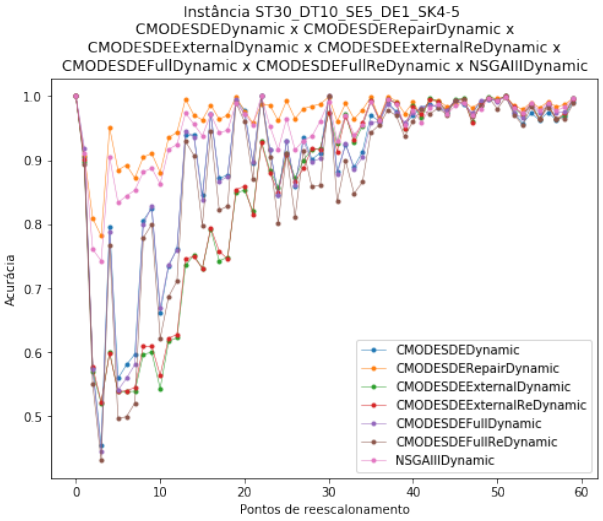

Figura B.32 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I13

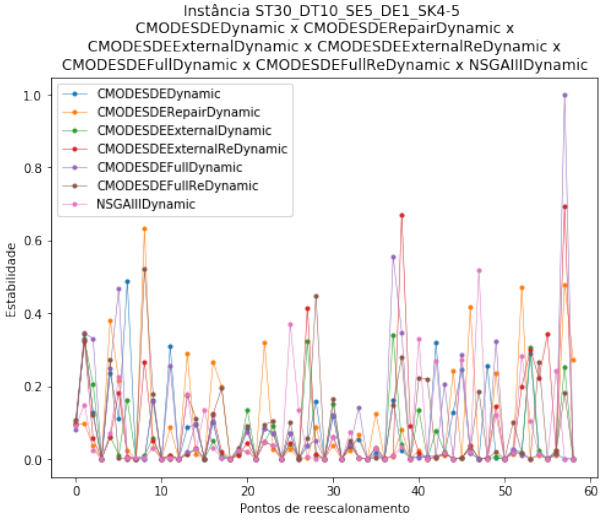

Figura B.33 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I13

B.12 Instância I14 (`sT30_dT10_sE10_dE1_SK4-5`)

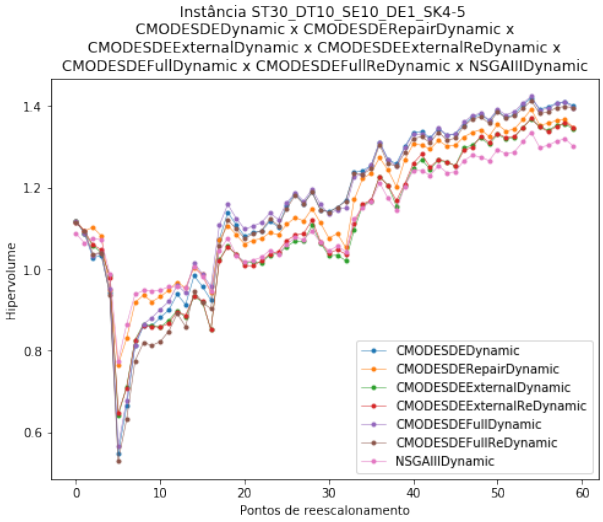

Figura B.34 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I14

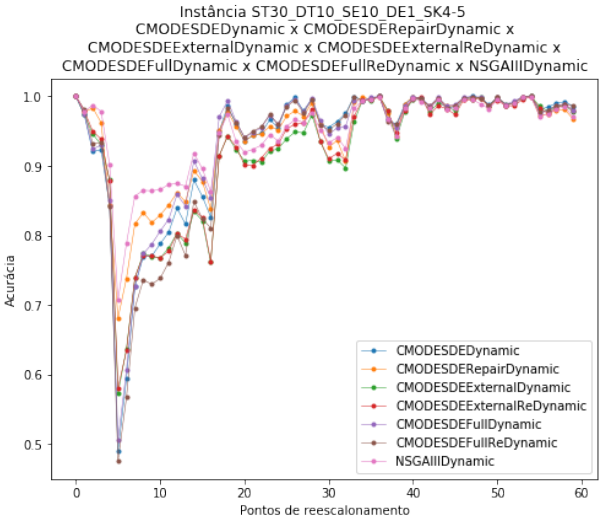

Figura B.35 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I14

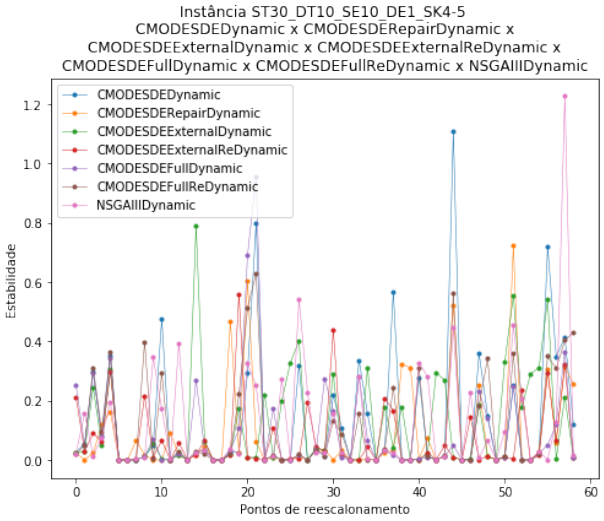

Figura B.36 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I14

B.13 Instância I15 (sT30_dT10_sE15_dE1_SK4-5)

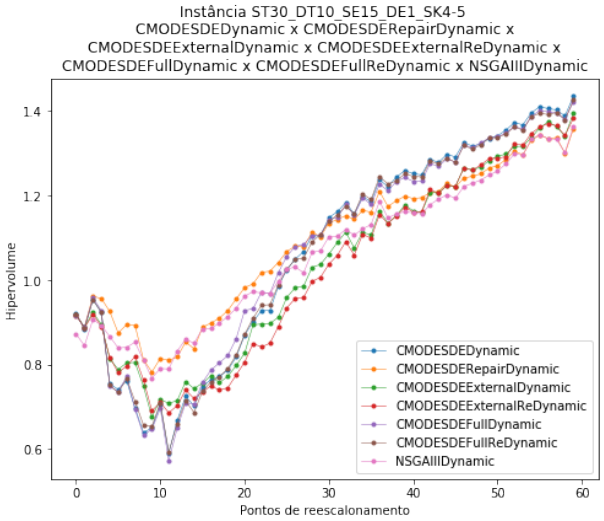

Figura B.37 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I15

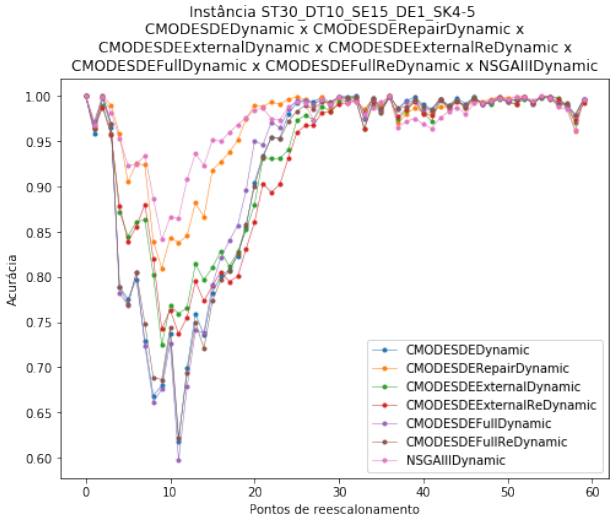

Figura B.38 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I15

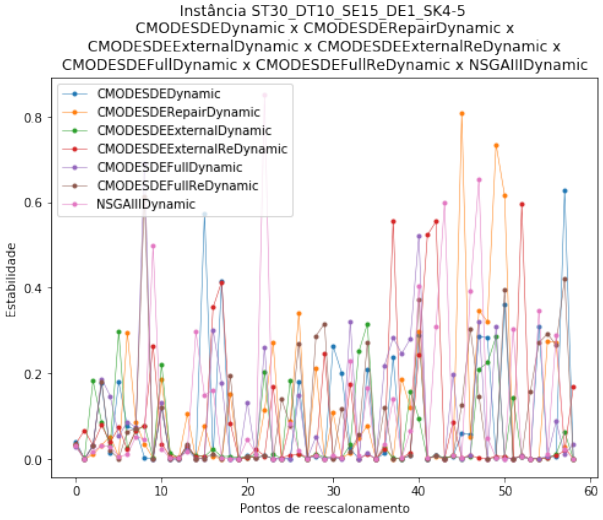

Figura B.39 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I15

B.14 Instância I16 (sT30_dT10_sE5_dE1_SK6-7)

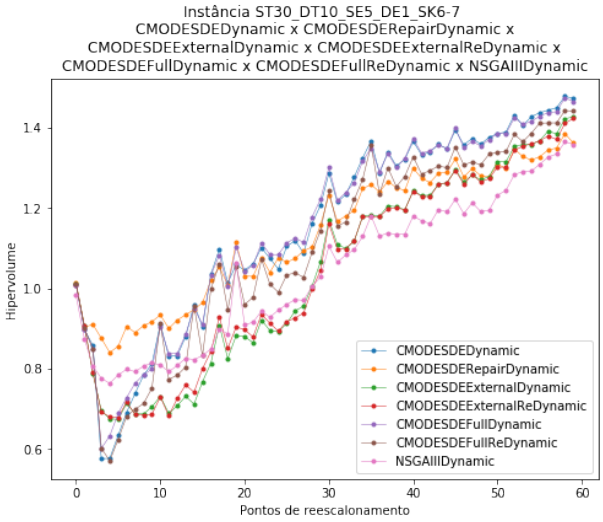

Figura B.40 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I16

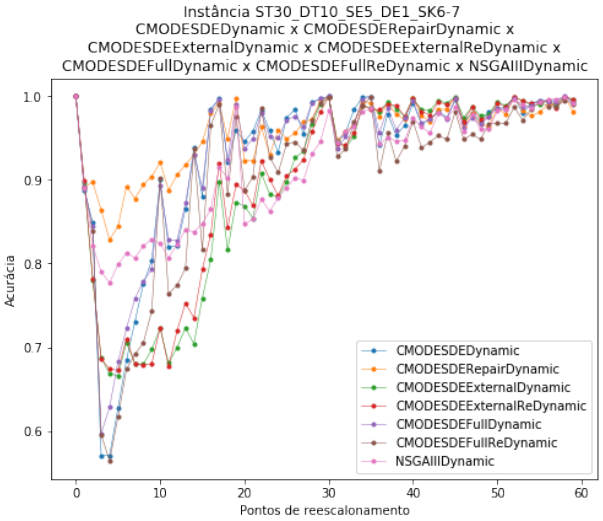

Figura B.41 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I16

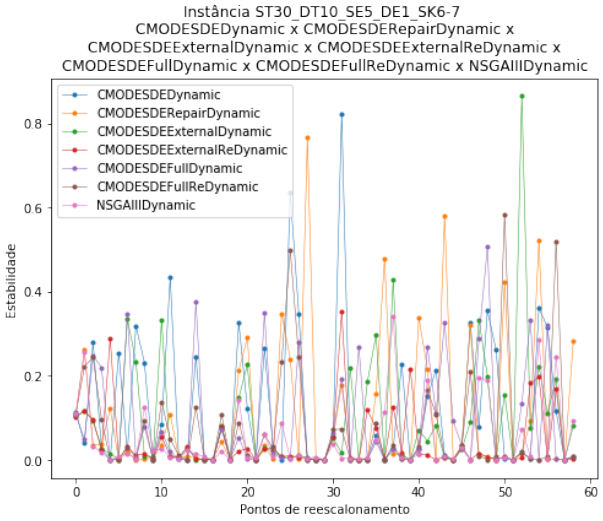

Figura B.42 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I16

B.15 Instância I18 (sT30_dT10_sE15_dE1_SK6-7)

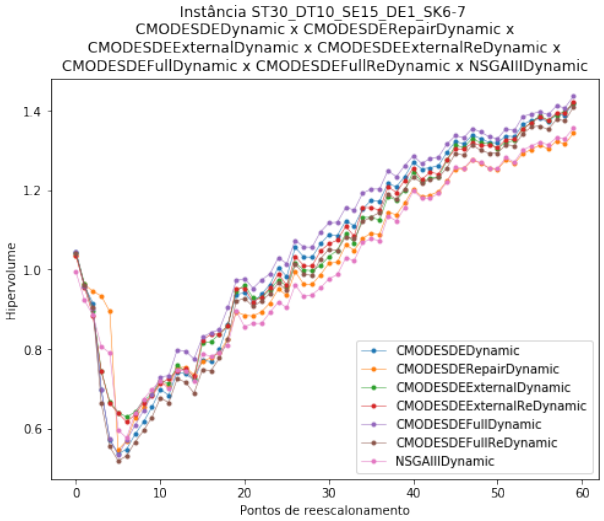

Figura B.43 – Comparação entre os hipervolumes em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I18

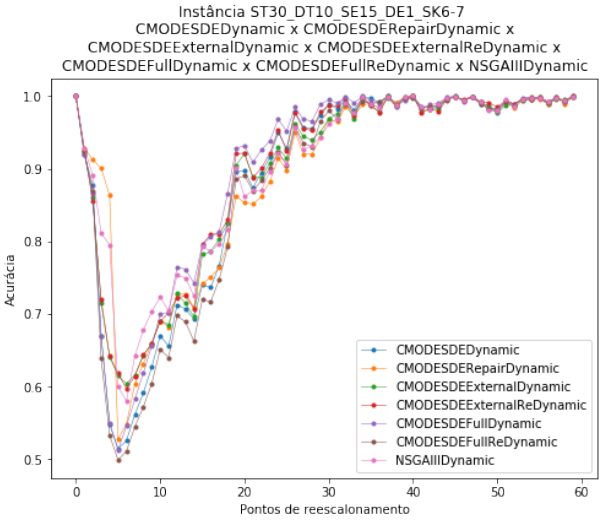

Figura B.44 – Comparação entre as acurácias em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I18

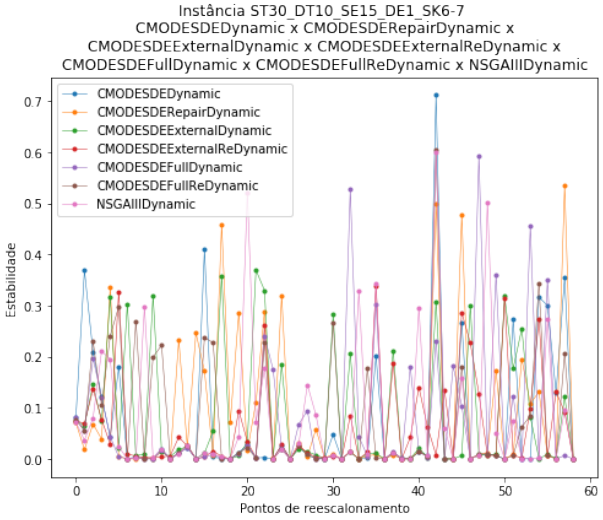

Figura B.45 – Comparação entre as estabilidades em cada ponto de reescalonamento para os algoritmos com técnicas dinâmicas para a instância I18